# Navigating the E4S Mathematical Solver Ecosystem: A Comparative Analysis for HPC Developers

## I. Introduction: Navigating the E4S Solver Ecosystem in the Exascale Era

The E4S 25.06 release represents a curated, integrated software ecosystem designed to address the central challenge of modern high-performance computing (HPC): hardware heterogeneity.1 The exascale landscape is defined by accelerator-based architectures 4, where "every vendor... has adopted its own programming model".4 E4S is the community's strategic response, providing a robust, tested software stack that supports diverse CPUs (x86\_64, ppc64le, aarch64) and, critically, GPUs from all major vendors: NVIDIA (via CUDA), AMD (via ROCm), and Intel (via oneAPI/SYCL).2

For application developers, this ecosystem presents a fundamental choice. The E4S math libraries portfolio, which includes foundational products like hypre, PETSc, SUNDIALS, SuperLU, and Trilinos 6, is not a flat collection of competing tools. Instead, it is a deliberately architected, two-tiered structure comprising:

1. **Comprehensive Frameworks (PETSc, Trilinos):** These are all-encompassing toolkits for building entire scientific applications. They provide integrated, high-level abstractions for the complete simulation workflow: mesh and discretization management, linear solvers, nonlinear solvers, and time integration.7
2. **Specialized Components (hypre, SuperLU, Ginkgo, SUNDIALS, etc.):** These are high-performance libraries that excel at a specific part of the problem. Examples include hypre for preconditioning 11, SuperLU for sparse direct solves 12, and Ginkgo for node-level sparse algebra.13

This structure is not accidental; it represents a "hub-and-spoke" model for scientific software. The frameworks (PETSc and Trilinos) serve as the "hubs," providing stable, long-term application-facing APIs. The specialized libraries (hypre, SuperLU, etc.) act as the high-performance, swappable "spokes." This design is proven by the libraries' interoperability: a user can, for example, build their application against the PETSc framework but configure it at runtime to use hypre's BoomerAMG preconditioner.14 Similarly, PETSc can wrap and use Trilinos's ML preconditioner 17, and Trilinos can, in turn, wrap PETSc data structures.18

This report provides a comparative analysis of these key libraries to guide selection. It contrasts the two core frameworks (PETSc and Trilinos), details the trade-offs between iterative and direct solvers, and provides deep-dive comparisons of specialist libraries. The goal is to provide a clear decision-making map, starting with the foundational components of the E4S math stack.

**Table 1: E4S Mathematical Library Feature Matrix**

| **Library** | **Primary Function** | **Core Language** | **Parallel Model** |
| --- | --- | --- | --- |
| **PETSc** | PDE Toolkit, Solvers (Linear, Nonlinear, TS) | C | MPI+X (Distributed) |
| **Trilinos** | Multiphysics Framework, Solvers, Kernels | C++ | MPI+X (Distributed) |
| **SuperLU** | Sparse Direct Solver (LU Factorization) | C | MPI+X (Distributed) / Shared |
| **Ginkgo** | Node-Level Sparse Linear Algebra | C++ | Node-Parallel (CUDA, HIP, SYCL) |
| **hypre** | High-Performance Preconditioners (AMG) | C | MPI+X (Distributed) |
| **SUNDIALS** | Time Integration (ODE/DAE Solvers) | C | MPI+X (Distributed) |
| **STRUMPACK** | Sparse Direct Solver / Preconditioner | C++ | MPI+X (Distributed) |
| **libCEED** | High-Order Finite Element Operator Algebra | C | Node-Parallel (CUDA, HIP, SYCL) |
| **FFTX** | Fast Fourier Transform (FFT) | C/C++ | MPI+X (Distributed) |
| **MAGMA** | Dense Linear Algebra (LAPACK/BLAS) | C/C++ | Hybrid CPU/GPU (Node-Parallel) |

## II. The First Strategic Decision: Frameworks vs. Specialized Libraries

The user's first choice dictates the entire application architecture: whether to build *within* a comprehensive framework or to build a custom application that *calls* specialized components.

### The Frameworks: PETSc and Trilinos as "Operating Systems" for PDEs

The two largest projects, PETSc and Trilinos, are best understood as integrated platforms for solving complex, (non)linear, time-dependent problems modeled by Partial Differential Equations (PDEs).7

* **PETSc (Portable, Extensible Toolkit for Scientific Computation)** provides a comprehensive suite of objects that abstract the entire solution process. These include Vec (vectors), Mat (matrices), IS (index sets), DM (discretization management), KSP (Krylov subspace solvers), PC (preconditioners), SNES (nonlinear solvers), TS (time steppers), and TAO (optimization).7 The DM object is a particularly powerful abstraction, managing the complex mapping between mesh or grid data structures and the underlying linear algebra.7
* **Trilinos** is a "collection of software packages" 8 explicitly designed for large-scale, complex multiphysics applications.22 It provides an equivalent, though more modular, set of capabilities through its many packages, such as Tpetra for core linear algebra 28, Belos for Krylov solvers, Amesos2 for direct solvers 8, Ifpack2 and MueLu for preconditioning 30, NOX for nonlinear solvers 8, and Anasazi for eigenvalue problems.8

### The Specialists: High-Performance Components for Targeted Tasks

In contrast to the all-encompassing frameworks, specialist libraries are designed to perform one task exceptionally well, typically with a focus on performance on specific hardware.

* **Ginkgo** is the premier example of a modern specialist. Its design explicitly focuses on "node parallelism" 33 for sparse linear algebra.13 It is not a distributed-memory (MPI) library; it is a GPU-first library providing optimized kernels for SpMV, iterative solvers, and preconditioners *on a single node*.35
* **SuperLU** is a specialist in a single algorithm class: the direct solution of sparse linear systems via LU factorization.12 It is not an iterative solver. It is used when robustness is paramount and iterative methods fail.38 It has serial, shared-memory, and distributed-memory (SuperLU\_DIST) versions.12
* **hypre** is a specialist in "high performance preconditioners".11 While it contains its own Krylov solvers 11, its international reputation is built on its Algebraic Multigrid (AMG) implementation, BoomerAMG.41 It is almost universally used as a "spoke" plugged into the PETSc "hub".15
* **SUNDIALS** is a specialist for an entirely different problem class: the time integration of Ordinary Differential Equations (ODEs) and Differential-Algebraic Equations (DAEs).43

The choice between these models is a trade-off between scope and control. Adopting a framework like PETSc or Trilinos means buying into its application development philosophy, data structures, and program flow.10 This is a high-level decision to build an application *on top of* the framework. In contrast, a developer with an existing, custom-built MPI application would not adopt Trilinos just to solve a linear system. They would instead call a specialist library like Ginkgo or SuperLU to accelerate that single, specific component.47 The "hub-and-spoke" model offers a powerful compromise: use the PETSc "hub" to manage the application's complexity but call the SuperLU or hypre "spoke" to perform the most difficult mathematical computations.

## III. The Core Frameworks: A Comparative Analysis of PETSc and Trilinos

For developers building new applications, the most significant choice is between the two foundational frameworks: PETSc and Trilinos. Their differences are not just in features, but in core philosophy, programming model, and user experience.

### Philosophical and Architectural Differences: Runtime vs. Compile-Time

The choice between PETSc and Trilinos is effectively a choice between a C-based, runtime-centric worldview and a C++-based, compile-time-centric one. This single philosophical difference has profound, causal effects on the user experience.

* **PETSc** is built on a C-based API 20 that implements an object-oriented *design*.23 Its defining philosophy is **runtime composability**. A user can build an application and then, at runtime, select and compose solvers, preconditioners, and their parameters via command-line arguments.32 This makes it an unparalleled tool for rapid algorithmic prototyping and experimentation.7 Its conceptual strength for PDE applications lies in the DM (Data Management) object, which provides a critical abstraction layer between the user's physics (e.g., a structured grid) and the library's algebra.7
* **Trilinos** is a C++ "framework of packages".25 Its defining philosophy is **compile-time composability** and C++-native abstraction.22 It is frequently described as being "like LEGO bricks" 26, where developers select the packages they need (e.g., Belos for solvers, MueLu for multigrid) and connect them at compile time. This design heavily leverages C++ templates (e.g., the Tpetra linear algebra package 50) and abstract interfaces (e.g., Thyra 50) to achieve its flexibility.

### The User and Developer Experience: A Stark Contrast

The downstream effects of these two philosophies are most apparent in user-reported experiences regarding documentation, community support, and build complexity.

* **Documentation and Community Support:** There is a strong consensus in user feedback that PETSc provides a superior out-of-box experience.
  + **PETSc** is described as the clear winner in documentation, with an "active and responsive mailing list," "a plethora of examples for most things," and "stellar" community support.46 The PETSc DM object is specifically praised as "much easier to use and well documented".46
  + **Trilinos**, conversely, is often criticized for its documentation. Users report that "the more niche you get the less documented it seems to be" and note its "poor community support".46 A common observation is that the older, now-deprecated packages (Epetra, AztecOO) are well-documented, but the modern, preferred packages (Tpetra, Belos) are "less documented".46 This is a natural consequence of its decentralized "framework of packages" model 26, which contrasts with PETSc's more unified design.
* **Build and Integration Complexity:** This is the single greatest point of friction for Trilinos.
  + **Trilinos**'s "LEGO brick" philosophy, with over 60 packages 26, necessitates an extremely complex build system (TriBITS, built on CMake) to manage the web of inter-package dependencies.53 This leads directly to user complaints that "the trilinos installer is nowhere documented, and the joke is that finding out what options are compatible in NP-hard".52 This is substantiated by documented build failures and complex linking errors.55
  + **PETSc**, while its configure.py script is massive, is generally seen as a more robust and self-contained build system that "works on the whole well" and is superior at handling external dependencies and cross-compilation.52
* **Programming Language and API:**
  + **PETSc**'s C-based API is a key feature. It provides simple, stable, and first-class bindings for C, Fortran, and Python 20, making it the default choice for applications not written in C++. This API is also its most-criticized aspect, with some C++ developers deriding it as a "museum artifact" that "screams 1980'ies".52
  + **Trilinos** is C++-native and has fully embraced the **Kokkos** programming model for performance portability.22 This modern API is powerful but comes with a steep learning curve. Users report that Kokkos-based C++ code can be "exceedingly hard to read if you are unfamiliar with it".46

The choice is clear: PETSc is optimized for the application scientist and experimentalist, prioritizing runtime flexibility and ease of use (especially for C/Fortran users). Trilinos is optimized for the C++ framework developer and computer scientist, prioritizing compile-time flexibility and C++-native abstractions for building large-scale, coupled multiphysics codes.

## IV. The Second Strategic Decision: Iterative vs. Direct Solvers

After selecting a framework (or deciding to use a specialist), the next critical decision is the solver algorithm class. This choice is a fundamental trade-off between computational scalability and numerical robustness.

* **Direct Solvers:** These libraries (e.g., **SuperLU**, **STRUMPACK**) are based on Gaussian elimination, computing an $LU$ factorization of the matrix $A$.12
  + **Pros:** They are "robust for difficult problems" 38 and "very accurate".57 They are the solver of last resort when iterative methods fail to converge, which is common for highly ill-conditioned or indefinite systems.19
  + **Cons:** They are computationally expensive and memory-intensive.57 Their time complexity scales poorly with problem size $N$ (e.g., $O(N^2)$ for 2D PDEs, $O(N^3)$ for 3D) 60, and they suffer from "fill-in," where the $L$ and $U$ factors become much denser than the original sparse matrix $A$.58
* **Iterative Solvers:** These libraries (e.g., **PETSc KSP**, **Trilinos Belos**, **Ginkgo**, **hypre**) use Krylov methods (e.g., GMRES, Conjugate Gradient) to find a solution by refining an initial guess.7
  + **Pros:** They are the *only* viable option for "very large sparse linear systems".11 When combined with an effective preconditioner like multigrid, their computational complexity can be "optimal," or $O(N)$.64 They consume far less memory than direct solvers.57
  + **Cons:** Convergence is *not* guaranteed.19 Their performance is entirely dependent on the quality of the preconditioner.63 An iterative solver that does not converge is useless.

In the past, this was a binary choice: direct for small/hard problems, iterative for large/easy problems. The modern HPC approach, however, blurs this line. Direct solvers are now frequently used as components *within* iterative methods. For example, SuperLU and STRUMPACK are explicitly designed to be used as "coarse grid solvers in a multigrid solver" or "subdomain solvers in a domain decomposition solver" 66—both of which are preconditioning techniques. Furthermore, libraries like STRUMPACK now use low-rank compression (BLR, HSS) to compute *approximate* factorizations.67 An approximate direct solve *is* a preconditioner. PETSc directly enables this dual-use case, allowing a user to select STRUMPACK as an exact direct solver (-pc\_type lu) or as an incomplete/approximate preconditioner (-pc\_type ilu).70

## V. Deep Dive: A Comparative Guide to E4S Solvers and Preconditioners

### Direct Solvers: SuperLU vs. STRUMPACK

* **SuperLU (Supernodal LU):** This is the E4S library of choice for the direct solution of *nonsymmetric* linear systems.12 Its algorithm is "supernodal," meaning it groups columns with similar sparsity patterns into "supernodes".60 This allows the algorithm to use highly efficient dense BLAS3 (matrix-matrix) operations, which reduces memory-access overheads.60 SuperLU is available in three distinct packages for different scales: SuperLU (serial), SuperLU\_MT (shared-memory parallel via Pthreads/OpenMP), and SuperLU\_DIST (distributed-memory parallel via MPI).12 The 3D algorithm in SuperLU\_DIST is noted for its enhanced scalability.73
* **STRUMPACK (STRUctured Matrix PACKage):** This library is designed for sparse systems that have a *symmetric pattern* (though the values may be nonsymmetric).68 It uses a *multifrontal* algorithm, which is "Tree based" 60 and exposes a different form of parallelism than supernodal methods.33 STRUMPACK's most important modern feature is its ability to use *rank-structured compression* (e.g., Block Low-Rank (BLR) and Hierarchically Semi-Separable (HSS)).67 This allows it to compress the "fill-in" during factorization, turning it into a "fast and scalable... approximate direct solver or preconditioner".69

### Algebraic Multigrid (AMG) Preconditioners: hypre vs. Trilinos (MueLu) vs. PETSc (PCGAMG)

For large-scale iterative solves, an Algebraic Multigrid (AMG) preconditioner is often the only scalable option. The E4S ecosystem provides three world-class, competing implementations.

* **hypre (BoomerAMG):** This is the *de facto* external standard for AMG. BoomerAMG is a parallel implementation of *classical* (Ruge-Stüben) AMG.14 It operates as a "black box" on the matrix $A$. Its key feature is its set of "conceptual interfaces" (Struct, SStruct).77 If a user's problem has a logically rectangular grid, they can use these "grey box" interfaces 11 to provide grid information to hypre, resulting in a much more efficient and scalable solver than the standard "black box" (IJ) interface.78 The most common use case is calling BoomerAMG from within PETSc, which has a well-documented interface with a vast number of tuning options.14
* **Trilinos (MueLu) and PETSc (PCGAMG):** These are the native AMG implementations within the two major frameworks. Both are based on a different mathematical approach: *aggregation-based* AMG.14 This approach uses "smoothed aggregation" to define the coarse grids.75 MueLu is the modern Trilinos multigrid package, built natively on Tpetra and Kokkos for GPU portability.22 PCGAMG is PETSc's native, modular implementation.80

The competition and interoperability in the AMG space perfectly illustrate the E4S "hub-and-spoke" strategy. A PETSc user, from a single common application code, can use the standard PC (preconditioner) interface 15 to choose between:

1. -pc\_type gamg: PETSc's *native* aggregation-based AMG.80
2. -pc\_type hypre: hypre's *external* classical AMG (requires --download-hypre).14
3. -pc\_type ml: Trilinos's *external* aggregation-based AMG (requires --download-ml).14

This allows a user to benchmark three different, world-class AMG algorithms on their specific problem by changing a single command-line flag. The E4S ecosystem fosters this competition (e.g., MueLu vs. hypre 84) while ensuring interoperability, with the user as the ultimate beneficiary.

### Node-Level Sparse Linear Algebra: Ginkgo vs. MAGMA-sparse

* **Ginkgo:** This is the modern, C++17, GPU-first library for *node-level* sparse linear algebra.13 It is not an MPI library. It provides the high-performance building blocks—SpMV, iterative solvers, preconditioners 35—optimized for manycore systems.37 It also provides a Python interface, pyGinkgo.47
* **MAGMA-sparse:** This library is now in "Legacy Support Mode".88 The official documentation "encouraged [users] to consider switching to the Ginkgo library".88 This is not a failure, but a sign of a healthy, actively managed ecosystem. The same development lead oversaw both MAGMA-sparse and Ginkgo 90, indicating a planned succession. MAGMA-sparse was the testbed for GPU sparse algorithms 91, and Ginkgo is the modern, from-scratch, performance-portable implementation of those ideas.47

## VI. Deep Dive: Domain-Specific E4S Math Libraries

Beyond the core linear solvers, the E4S 25.06 release includes several other critical, domain-specific libraries.

### SUNDIALS: The Specialist for Time-Dependent Problems (ODEs/DAEs)

SUNDIALS (SUite of Nonlinear and DIfferential/ALgebraic Equation Solvers) is the primary E4S library for time integration. It provides a suite of solvers for Ordinary Differential Equations (ODEs) (e.g., CVODE, ARKODE) and Differential-Algebraic Equations (DAEs) (e.g., IDA).43 It also includes KINSOL for nonlinear algebraic systems.43

While PETSc also offers a time-stepping module (TS) 7, the libraries are both competitors and collaborators. PETSc provides an interface, TSSUNDIALS, to use the SUNDIALS CVODE solver as a backend.93 A user might choose SUNDIALS directly for its more extensive collection of methods (e.g., BDF, Adams-Moulton) and its advanced capabilities for forward and adjoint sensitivity analysis.43 SUNDIALS's flexible design, built on abstract interfaces for vectors and matrices 43, has enabled its effective porting to GPUs 96 and its support for portability layers like Kokkos and RAJA.98

### libCEED: The "Post-Matrix" Library for High-Order Methods

libCEED is a unique and forward-looking library. It is *not* a solver. It is a library that provides "performance-portable *algebra* for element-based discretizations".101

This library exists to solve a critical performance bottleneck in modern simulations. For high-order finite element methods (FEM), "a global sparse matrix is no longer a good representation of a high-order linear operator".102 The computational cost and memory bandwidth required to assemble this large sparse matrix become the dominant bottleneck, not the solve itself.

libCEED's solution is to provide a "matrix-free" or "algebraically factored" format.102 The user describes the *operator* algebraically—by defining its basis evaluators and action at quadrature points.104 libCEED then provides highly optimized, architecture-specific backends (e.g., pure C, AVX, CUDA, HIP, SYCL) 102 that can *apply* this operator without ever forming the global matrix. libCEED is designed to be a "spoke" that sits *underneath* frameworks like PETSc and MFEM.104 A user would employ a PETSc KSP solver with a Mat-Shell (a matrix-free operator), where the "apply" routine is simply a highly-optimized call to libCEED.

### FFTX: The Next-Generation FFT Library

FFTX is the E4S specialist for Fast Fourier Transforms.2 It is the designated exascale follow-on to the ubiquitous FFTW library.108 This succession is necessary because "FFTW is no longer under active development" and its design does not map well to "complex heterogeneous composites of new types of processors" (i.e., GPUs).108

FFTX's core technology is the SPIRAL code generation system.108 This allows FFTX to generate "high-performance kernels targeted to specific uses and platform environments".108 This is a higher-level approach; instead of just providing an FFT, the API allows users to specify entire FFT-based kernels (e.g., gather-FFT-operate-iFFT-scatter), which SPIRAL then compiles into a single, optimized, hardware-independent function.108

### MAGMA (Dense): The LAPACK/BLAS for Hybrid Architectures

While MAGMA-sparse is deprecated, the main **MAGMA (Matrix Algebra on GPU and Multi-core Architectures)** library is the E4S solution for *dense* linear algebra.88 It provides LAPACK-compliant routines for hybrid CPU/GPU systems.88 Its core design is "hybrid CPU-GPU execution," which intelligently splits tasks between the CPU and GPU to optimize resource utilization.113 This contrasts with vendor libraries like cuBLAS, which are often GPU-only.113 MAGMA is a C library 115 with CPU- and GPU-resident interfaces 111 and is also accessible via Python through PyMAGMA.115

## VII. The Critical Axis: GPU Portability in the E4S Ecosystem

For modern applications, the most complex and critical decision is the GPU portability strategy. The E4S libraries demonstrate three distinct and competing philosophies for supporting NVIDIA, AMD, and Intel GPUs.

### Model 1: The "Write-Once" Portability Layer (Trilinos & Kokkos)

Trilinos has made a strategic, all-in commitment to the **Kokkos** programming model.22 Kokkos is a C++ library that provides abstractions for both parallel execution (e.g., parallel\_for, parallel\_reduce) and data structures (e.g., Kokkos::View).120 Developers write their algorithms *once* using the Kokkos C++ API. At compile time, Kokkos maps these algorithms to the appropriate hardware backend, whether it be CUDA, HIP, SYCL, or OpenMP.120 Key Trilinos packages, including Tpetra and MueLu, are built this way.22 The benefit is a single-source C++ codebase for all architectures.22 The cost is a commitment to the Kokkos ecosystem, which has a steep learning curve 46, and performance is gated by the maturity of the Kokkos backend for each specific architecture.

### Model 2: The "Flexible Abstraction" (PETSc & SUNDIALS)

This is a more flexible, hybrid approach. These libraries *support* portability layers but are not *exclusive* to them.

* **PETSc**'s design "stresses flexibility and extensibility" and "separat[es] the programming model used by the application from that used by the library".125 This allows an application to be written in Kokkos, while PETSc itself might be configured to use its native, hand-tuned CUDA or HIP kernels.127 PETSc supports *both* native backends (CUDA, HIP, OpenCL) and portability abstractions (Kokkos, RAJA, SYCL).125
* **SUNDIALS** follows the same philosophy. It provides native support for CUDA, HIP, and SYCL 96, but *also* provides interfaces for applications using Kokkos 98 and RAJA.96

This model provides maximum user flexibility and future-proofing, but at the cost of a more complex API and build configuration.

### Model 3: The "Native Backend" (Ginkgo, libCEED, SuperLU, hypre)

This model separates the high-level, hardware-agnostic algorithm from a set of discrete, hand-optimized, hardware-specific backends.

* **Ginkgo** is the premier example of this model. Its abstract C++ algorithms 47 call architecture-specific kernels written natively in CUDA, HIP, and SYCL/DPC++.13
* **libCEED**, **hypre**, and **SuperLU\_DIST** follow the same design, maintaining separate, native implementations for CUDA, HIP, and (for libCEED and hypre) SYCL.12

The advantage of this model is the potential for the highest, "to-the-metal" performance by using vendor-native languages and libraries (e.g., cuBLAS, rocBLAS) directly.36 The disadvantage is a massive developer burden to write and maintain $N$ separate code paths.135 This trade-off is so significant that even the PETSc team, which uses this model, has called it "painful to maintain duplicated code" and is now adopting Kokkos for *some* of its components.4

### Maturity of CUDA, ROCm, and SYCL Support

* **CUDA (NVIDIA):** Universally mature. This is the baseline for performance and features, with long-standing support across all major libraries.13
* **ROCm (AMD):** Widespread support, but with real-world caveats. All major libraries (PETSc, Trilinos, Ginkgo, hypre, SUNDIALS, STRUMPACK, libCEED) have HIP backends.13 The CUDA-to-HIP porting process is a common, semi-automated task.130 However, the user-facing ecosystem can be less stable than CUDA, with reports of installation issues.142
* **SYCL (Intel):** This is the newest and most rapidly evolving backend. Adoption is fast: Ginkgo 13, libCEED 102, hypre 132, and SUNDIALS 96 all have SYCL backends. STRUMPACK has a functional SYCL implementation with performance tuning ongoing.138 PETSc's support is still listed as "in development" 127, reflecting the emerging status of this backend.

**Table 2: GPU Portability and Programming Model Comparison**

| **Library** | **Portability Strategy** | **CUDA Support** | **ROCm (HIP) Support** | **SYCL Support** |
| --- | --- | --- | --- | --- |
| **PETSc** | **Flexible Abstraction** (Native + Portability Layers) | Mature 127 | Mature 127 | In Development 127 |
| **Trilinos** | **"Write-Once" Layer** (Kokkos) | Mature 136 | Mature 136 | Mature 136 |
| **SuperLU** | **Native Backend** | Mature 12 | Mature 72 | No |
| **Ginkgo** | **Native Backend** | Mature 13 | Mature 13 | Mature 13 |
| **hypre** | **Native Backend** | Mature 132 | Mature 132 | Mature 132 |
| **SUNDIALS** | **Flexible Abstraction** (Native + Portability Layers) | Mature 98 | Mature 98 | Mature 96 |
| **STRUMPACK** | **Native Backend** (via SLATE, MAGMA, etc.) | Mature 137 | Mature 138 | Functional 138 |
| **libCEED** | **Native Backend** | Mature 102 | Mature 102 | Mature 102 |

## VIII. Synthesis and Recommendations: Selecting the Right Library Stack for Your Problem

The E4S ecosystem is designed to be interoperable. The optimal "choice" is often a *stack* of libraries (e.g., PETSc + hypre + libCEED). The following recommendations are based on common user scenarios.

### Scenario 1: Starting a new, large-scale multiphysics application in C++.

* **Recommendation: Trilinos.**
* **Rationale:** Trilinos was *designed* for this. Its C++ "framework of packages" 25 and object-oriented abstractions 50 are built for coupling multiphysics components.22 Its complete adoption of the Kokkos programming model 22 provides a compelling, single-source C++ portability solution for CPUs and all major GPU vendors.136 Be prepared to invest significant time in learning the Kokkos model 46 and navigating the complex TriBITS build system.52

### Scenario 2: Modernizing a legacy PDE-based application (C or Fortran).

* **Recommendation: PETSc.**
* **Rationale:** PETSc's stable C API, with first-class Fortran and Python bindings 20, is the ideal on-ramp. Its superior documentation, extensive examples, and "stellar" community support 46 are invaluable for reducing developer friction. Its runtime composability 32 allows scientists to experiment with solvers without recompiling. Finally, its "Flexible Abstraction" model for GPUs 125 provides the most gradual and lowest-risk path to accelerator-based computing.

### Scenario 3: Your iterative solver fails to converge (ill-conditioned or indefinite system).

* **Recommendation: SuperLU\_DIST (for nonsymmetric) or STRUMPACK (for symmetric-pattern).**
* **Rationale:** When iterative solvers fail, a robust direct solver is required.19
  + Choose **SuperLU\_DIST** if your matrix is *nonsymmetric*.12
  + Choose **STRUMPACK** if your matrix has a *symmetric pattern*.68 Critically, investigate using STRUMPACK with its low-rank compression (BLR/HSS) enabled 69, which functions as a "fast and scalable... approximate direct solver or preconditioner" 69 for your Krylov method.

### Scenario 4: Writing a new, high-performance, ***node-level*** (single-node, multi-GPU) kernel in C++.

* **Recommendation: Ginkgo.**
* **Rationale:** This is Ginkgo's explicit design goal.13 It is a modern C++17, GPU-first library with native, hand-optimized backends for CUDA, HIP, and SYCL.37 It is also the officially-endorsed successor to the legacy MAGMA-sparse library.88 Do not choose PETSc or Trilinos for this task; as distributed-memory (MPI) frameworks, they are massive overkill.

### Scenario 5: You need the ***absolute best*** Algebraic Multigrid (AMG) preconditioner for a large MPI job.

* **Recommendation: hypre (BoomerAMG), called from PETSc.**
* **Rationale:** hypre is the E4S specialist library for high-performance preconditioners.11 BoomerAMG is its flagship classical AMG product.42 The most common, well-documented, and robust path is to use PETSc as the application framework and call hypre via the simple -pc\_type hypre runtime flag.15 This stack gives you the best-in-class solver component (hypre) with the best-in-class user framework (PETSc).

### Scenario 6: Your application is dominated by time integration (ODEs/DAEs).

* **Recommendation: SUNDIALS.**
* **Rationale:** This is SUNDIALS's sole focus.43 It provides a more extensive and advanced suite of time-stepping algorithms (e.g., BDF, ARKODE) and sensitivity analysis capabilities than the native PETSc TS module.43 Its flexible design 43 has allowed it to be ported effectively to GPUs with support for multiple backends.96

### Scenario 7: You are developing a ***high-order*** (e.g., **$Q\_k$** with **$k \ge 2$**) Finite Element Method (FEM) code.

* **Recommendation: libCEED (for operators) + PETSc or Trilinos (for solvers).**
* **Rationale:** For high-order methods, the cost of *assembling* the global sparse matrix is a dominant bottleneck.102 libCEED is designed to solve this problem by providing a "matrix-free" / "factored" operator algebra.101 The optimal, modern stack is to use libCEED to handle the finite element operator *application* and to feed this operator into a PETSc KSP or Trilinos Belos solver, which will solve the system matrix-free.104

### Scenario 8: Your application is dominated by Fast Fourier Transforms (FFTs).

* **Recommendation: FFTX.**
* **Rationale:** The previous standard, FFTW, is no longer actively developed and is not designed for heterogeneous GPU architectures.108 FFTX is the designated E4S successor 3, built on the SPIRAL code-generation engine to produce highly optimized, platform-specific kernels for modern hardware.108

#### Works cited

1. E4S - Home, accessed November 6, 2025, <https://e4s.io/>
2. NEWS RELEASE - E4S, accessed November 6, 2025, <https://e4s.io/news/NEWS_RELEASE_E4S_25.06.pdf>
3. E4S: The Extreme-scale Scientific Software Stack Release 25.06, accessed November 6, 2025, <https://oaciss.uoregon.edu/e4s/talks/E4S_25.06.pdf>
4. Optimizing PETSc for exascale | Argonne Leadership Computing Facility, accessed November 6, 2025, <https://www.alcf.anl.gov/news/optimizing-petsc-exascale>
5. E4S: The Extreme-scale Scientific Software Stack Release 24.11, accessed November 6, 2025, <https://oaciss.uoregon.edu/e4s/talks/E4S_24.11.pdf>
6. Overview of Numerical Algorithms and Software for Extreme-Scale Science, accessed November 6, 2025, <https://wordpress.cels.anl.gov/atpesc/wp-content/uploads/sites/96/2020/08/ATPESC-2020-Track-5-Talk-1-McInnesAndMiller-IntroToNumericalSoftware-v3-1.pdf>
7. PETSc Users Manual - Argonne Scientific Publications, accessed November 6, 2025, <https://publications.anl.gov/anlpubs/2016/05/127241.pdf>
8. Trilinos Tutorial, accessed November 6, 2025, <https://trilinos.github.io/pdfs/Trilinos8.0Tutorial.pdf>
9. Portable, Extensible Toolkit for Scientific Computation - Wikipedia, accessed November 6, 2025, <https://en.wikipedia.org/wiki/Portable,_Extensible_Toolkit_for_Scientific_Computation>
10. What are the main differences between PETSc and Trilinos?, accessed November 6, 2025, <https://scicomp.stackexchange.com/questions/461/what-are-the-main-differences-between-petsc-and-trilinos>
11. hypre Documentation, accessed November 6, 2025, <https://hypre.readthedocs.io/_/downloads/en/latest/pdf/>
12. SuperLU: Home Page - NERSC, accessed November 6, 2025, <https://portal.nersc.gov/project/sparse/superlu/>
13. Ginkgo, accessed November 6, 2025, <https://ginkgo-project.github.io/>
14. Algebraic multigrid in PETSc - Computational Science Stack Exchange, accessed November 6, 2025, <https://scicomp.stackexchange.com/questions/27359/algebraic-multigrid-in-petsc>
15. KSP: Linear System Solvers — PETSc v3.24.0-158-gdafee7125fca documentation, accessed November 6, 2025, <https://petsc.org/main/manual/ksp/>
16. PETSc+Hypre - Forums - CFD Online, accessed November 6, 2025, <https://www.cfd-online.com/Forums/main/8917-petsc-hypre.html>
17. Supported External Software — PETSc 3.24.1 documentation, accessed November 6, 2025, <https://petsc.org/release/install/external_software/>
18. EpetraExt: Trilinos/PETSc Interface, accessed November 6, 2025, <https://docs.trilinos.org/dev/packages/epetraext/doc/html/epetraext_petsc_interface.html>
19. trilinos or petsc,which is better? - Forums - CFD Online, accessed November 6, 2025, <https://www.cfd-online.com/Forums/main/62946-trilinos-petsc-better.html>
20. PETSc — PETSc 3.24.1 documentation, accessed November 6, 2025, <https://petsc.org/>
21. PETSc/TAO Users Manual, accessed November 6, 2025, <https://petsc.org/release/manual/manual.pdf>
22. Trilinos: Enabling Scientific Computing Across Diverse Hardware Architectures at Scale - arXiv, accessed November 6, 2025, <https://arxiv.org/html/2503.08126v1>
23. Getting Started — PETSc 3.24.1 documentation, accessed November 6, 2025, <https://petsc.org/release/manual/getting_started/>
24. PETSc/TAO - Exascale Computing Project, accessed November 6, 2025, <https://www.exascaleproject.org/research-project/petsc-tao/>
25. Primary repository for the Trilinos Project - GitHub, accessed November 6, 2025, <https://github.com/trilinos/Trilinos>
26. Trilinos Progress, Challenges and Future Plans, accessed November 6, 2025, <https://trilinos.github.io/pdfs/HerouxTrilinosProgressChallengesFutures.pdf>
27. Helping Large-Scale Multiphysics Engineering and Scientific Applications Achieve Their Goals - Exascale Computing Project, accessed November 6, 2025, <https://www.exascaleproject.org/helping-large-scale-multiphysics-engineering-and-scientific-applications-achieve-their-goals/>
28. Kokkos Applications, accessed November 6, 2025, <https://kokkos.org/community/applications/>
29. Trilinos Progress, Challenges and Future Plans, accessed November 6, 2025, <https://trilinos.github.io/pdfs/3-HerouxTrilinosProgressChallengesFutures.pdf>
30. MueLu - Trilinos, accessed November 6, 2025, <https://trilinos.github.io/muelu.html>
31. MueLu User's Guide - Trilinos, accessed November 6, 2025, <https://trilinos.github.io/pdfs/mueluguide.pdf>
32. Introduction to Nonlinear Solvers Using PETSc/TAO - ATPESC, accessed November 6, 2025, <https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2022/11/ATPESC-2022-Track-5-Talk-5-RichardTranMills-NonlinearSolversUsingPETSc.pdf>
33. Preparing sparse solvers for exascale computing - PMC - PubMed Central - NIH, accessed November 6, 2025, <https://pmc.ncbi.nlm.nih.gov/articles/PMC7015295/>
34. ginkgo-project/ginkgo: Numerical linear algebra software package - GitHub, accessed November 6, 2025, <https://github.com/ginkgo-project/ginkgo>
35. Ginkgo Documentation, accessed November 6, 2025, <https://ginkgo-project.github.io/ginkgo-generated-documentation/doc/develop/>
36. Sparse Linear Algebra on AMD and NVIDIA GPUs – The Race Is On - PubMed Central - NIH, accessed November 6, 2025, <https://pmc.ncbi.nlm.nih.gov/articles/PMC7295357/>
37. Ginkgo and oneAPI Accelerate Numerical Simulations on Intel® GPUs, accessed November 6, 2025, <https://www.intel.com/content/www/us/en/developer/articles/technical/ginkgo-and-oneapi-accelerate-numerical-simulations.html>
38. Sparse, Direct Solvers with SuperLU, accessed November 6, 2025, <https://xsdk-project.github.io/MathPackagesTraining2021/lessons/superlu_mfem/>
39. SuperLU DIST: A Scalable Distributed-Memory Sparse Direct Solver for Unsymmetric Linear Systems\* Xiaoye S. Li† James W. Demmel - OSTI.GOV, accessed November 6, 2025, <https://www.osti.gov/servlets/purl/836786>
40. hypre-space/hypre: Parallel solvers for sparse linear systems featuring multigrid methods. - GitHub, accessed November 6, 2025, <https://github.com/hypre-space/hypre>
41. Hypre BoomerAMG - MOOSE framework, accessed November 6, 2025, <https://mooseframework.inl.gov/application_usage/hypre.html>
42. Iterative Solvers and Algebraic Multigrid (with HYPRE) - Argonne Training Program on Extreme-Scale Computing, accessed November 6, 2025, <https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2023/08/ATPESC-2023-Track-5-Talk-3-OsbornYang-Iterative-Solvers-hypre.pdf>
43. SUNDIALS — User Documentation for SUNDIALS documentation, accessed November 6, 2025, <https://sundials.readthedocs.io/>
44. SUNDIALS: SUite of Nonlinear and DIfferential/ALgebraic Equation Solvers - | Computing, accessed November 6, 2025, <https://computing.llnl.gov/projects/sundials>
45. Official development repository for SUNDIALS - a SUite of Nonlinear and DIfferential/ALgebraic equation Solvers. Pull requests are welcome for bug fixes and minor changes. - GitHub, accessed November 6, 2025, <https://github.com/LLNL/sundials>
46. Any users of Trilinos? How does it compare to PETc? : r/fea - Reddit, accessed November 6, 2025, <https://www.reddit.com/r/fea/comments/zuwuyy/any_users_of_trilinos_how_does_it_compare_to_petc/>
47. pyGinkgo: A Sparse Linear Algebra Operator Framework For Python - arXiv, accessed November 6, 2025, <https://arxiv.org/html/2510.08230v1>
48. On the way towards an accelerator-focused C++ sparse linear algebra library, accessed November 6, 2025, <https://icl.utk.edu/~hanzt/talks/ginkgo_tutorial_2018.pdf>
49. Performance Analysis and Engineering in TOPS, accessed November 6, 2025, <https://www.mcs.anl.gov/research/projects/performance/tops.htm>
50. An Overview of Trilinos, accessed November 6, 2025, <https://trilinos.github.io/pdfs/TrilinosOverview.pdf>
51. Ge#ng Started with Trilinos, accessed November 6, 2025, <https://trilinos.github.io/pdfs/1-GettingStartedWithTrilinos.pdf>
52. Looking for advices : r/HPC - Reddit, accessed November 6, 2025, <https://www.reddit.com/r/HPC/comments/qclcti/looking_for_advices/>
53. Trilinos Configure, Build, Test, and Install Reference Guide, accessed November 6, 2025, <https://docs.trilinos.org/files/TrilinosBuildReference.html>
54. Trilinos Configure, Build, Test, and Install Reference Guide, accessed November 6, 2025, <https://trilinos.github.io/pdfs/TrilinosBuildReference.pdf>
55. Configuration problem with Trilinos from system libraries · Issue #17916 - GitHub, accessed November 6, 2025, <https://github.com/dealii/dealii/issues/17916>
56. Error in installing Trilinos - Google Groups, accessed November 6, 2025, <https://groups.google.com/g/dealii/c/L5yM2Cyj5SQ>
57. Linear solvers: Standard options - Altair Product Documentation, accessed November 6, 2025, <https://help.altair.com/flux/flux/help/english/UserGuide/English/topics/LinearSolversStandardOptions.htm>
58. SuperLU vs Direct Substructuring | Christoph Conrads' Blog, accessed November 6, 2025, <https://christoph-conrads.name/superlu-vs-direct-substructuring/>
59. Direct vs iterative solver - Ansys Customer Center, accessed November 6, 2025, <https://innovationspace.ansys.com/forum/forums/topic/direct-vs-iterative-solver/>
60. Direct Sparse Linear Solvers, Preconditioners, accessed November 6, 2025, <https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2023/08/ATPESC-2023-Track-5-Talk-3-Li-Ghysels-DirectSolvers.pdf>
61. linear algebra - Direct vs Iterative solvers choice - Mathematics Stack Exchange, accessed November 6, 2025, <https://math.stackexchange.com/questions/1842630/direct-vs-iterative-solvers-choice>
62. Summary of Sparse Linear Solvers Available In PETSc, accessed November 6, 2025, <https://petsc.org/release/overview/linear_solve_table/>
63. How to Choose a Solver for FEM Problems: Direct or Iterative? - SimScale, accessed November 6, 2025, <https://www.simscale.com/blog/how-to-choose-solvers-for-fem/>
64. ATPESC 2022 Krylov Solvers and Algebraic Multigrid with hypre - Argonne Training Program on Extreme-Scale Computing, accessed November 6, 2025, <https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2022/11/ATPESC-2022-Track-5-Talk-8-OsbornYang-Iterative-Solvers-hypre.pdf>
65. 35 PETSc solvers - The Art of HPC, accessed November 6, 2025, <https://theartofhpc.com/pcse/petsc-solver.html>
66. STRUMPACK/SuperLU - Exascale Computing Project, accessed November 6, 2025, <https://www.exascaleproject.org/research-project/strumpack-superlu/>
67. STRUMPACK | STRUctured Matrix PACKage, accessed November 6, 2025, <https://portal.nersc.gov/project/sparse/strumpack/>
68. STRUMPACK Speeds Sparse Algorithms on CPUs and GPUs - Exascale Computing Project, accessed November 6, 2025, <https://www.exascaleproject.org/highlight/strumpack-speeds-sparse-algorithms-on-cpus-and-gpus/>
69. STRUMPACK v7.2.0: Overview - NERSC, accessed November 6, 2025, <https://portal.nersc.gov/project/sparse/strumpack/master/>
70. MATSOLVERSTRUMPACK — PETSc 3.24.1 documentation, accessed November 6, 2025, <https://petsc.org/release/manualpages/Mat/MATSOLVERSTRUMPACK/>
71. Sparse LU for block-sparse matrices - Computational Science Stack Exchange, accessed November 6, 2025, <https://scicomp.stackexchange.com/questions/5348/sparse-lu-for-block-sparse-matrices>
72. Multi-GPU capabilities in SuperLU and STRUMPACK sparse direct solvers - Exascale Computing Project, accessed November 6, 2025, <https://www.exascaleproject.org/wp-content/uploads/2022/06/LiSherrySparseBofSlides.pdf>
73. An Alternate GPU-Accelerated Algorithm for Very Large Sparse LU Factorization - MDPI, accessed November 6, 2025, <https://www.mdpi.com/2227-7390/11/14/3149>
74. xiaoyeli/superlu\_dist: Distributed memory, MPI based SuperLU - GitHub, accessed November 6, 2025, <https://github.com/xiaoyeli/superlu_dist>
75. [petsc-users] Hypre's BoomerAMG vs PETSc's GAMG?, accessed November 6, 2025, <https://lists.mcs.anl.gov/pipermail/petsc-users/2015-September/027143.html>
76. BoomerAMG — hypre 3.0.0 documentation - Read the Docs, accessed November 6, 2025, <https://hypre.readthedocs.io/en/latest/solvers-boomeramg.html>
77. Documentation for hypre — hypre 3.0.0 documentation, accessed November 6, 2025, <https://hypre.readthedocs.io/>
78. Introduction — hypre 3.0.0 documentation - Read the Docs, accessed November 6, 2025, <https://hypre.readthedocs.io/en/latest/ch-intro.html>
79. Krylov Solvers and Algebraic Multigrid with hypre, accessed November 6, 2025, <https://xsdk-project.github.io/MathPackagesTraining2020/lessons/krylov_amg_hypre/>
80. KSP: Linear System Solvers — PETSc 3.19.6 documentation, accessed November 6, 2025, <https://web.cels.anl.gov/projects/petsc/vault/petsc-3.19.6/docs/manual/ksp.html>
81. ML | Trilinos, accessed November 6, 2025, <https://trilinos.github.io/ml.html>
82. What is a robust, iterative solver for large 3-d linear-elastic problems?, accessed November 6, 2025, <https://scicomp.stackexchange.com/questions/2369/what-is-a-robust-iterative-solver-for-large-3-d-linear-elastic-problems>
83. MueLu is a multigrid library in Sandia's Trilinos project and is designed to be flexible, easily extensible, and efficient on, accessed November 6, 2025, <https://www.orau.gov/scidac3pi2015/posters/Prokopenko_Andrey-MueLu_Multigrid_Framework.pdf>
84. A Comparison of Classical and Aggregation-Based Algebraic Multigrid Preconditioners for High-Fidelity Simulation of Wind Turbine Incompressible Flows - ResearchGate, accessed November 6, 2025, <https://www.researchgate.net/publication/336883831_A_Comparison_of_Classical_and_Aggregation-Based_Algebraic_Multigrid_Preconditioners_for_High-Fidelity_Simulation_of_Wind_Turbine_Incompressible_Flows>
85. PyAMG: Algebraic Multigrid Solvers in Python - Open Journals, accessed November 6, 2025, <https://www.theoj.org/joss-papers/joss.04142/10.21105.joss.04142.pdf>
86. Comparison of Some RANS Solvers (Conference) | OSTI.GOV, accessed November 6, 2025, <https://www.osti.gov/biblio/1883200>
87. Ginkgo and oneAPI accelerate numerical simulations on Intel GPUs, accessed November 6, 2025, <https://oneapi.io/blog/ginkgo-and-oneapi-accelerate-numerical-simulations-on-intel-gpus/>
88. MAGMA - Innovative Computing Laboratory - University of Tennessee, Knoxville, accessed November 6, 2025, <https://icl.utk.edu/magma/>
89. MAGMA: Sparse Overview, accessed November 6, 2025, <https://icl.utk.edu/projectsfiles/magma/doxygen/_m_a_g_m_a-sparse.html>
90. The Ginkgo Library Project is Developing a Vision for Multiprecision Algorithms, accessed November 6, 2025, <https://www.exascaleproject.org/the-ginkgo-library-project-is-developing-a-vision-for-multiprecision-algorithms/>
91. U.S. Department of Energy Office of Science ASYNCHRONOUS ITERATIVE SOLVERS FOR EXTREME-SCALE COMPUTING DOE-UTK-DE-SC0016513-1 Ja - OSTI.gov, accessed November 6, 2025, <https://www.osti.gov/servlets/purl/1764239>
92. TS: Scalable ODE and DAE Solvers — PETSc 3.24.1 documentation, accessed November 6, 2025, <https://petsc.org/release/manual/ts/>
93. PetSc vs Sundials for serial numerical computations?, accessed November 6, 2025, <https://scicomp.stackexchange.com/questions/24203/petsc-vs-sundials-for-serial-numerical-computations>
94. TSSUNDIALS — PETSc 3.23.6 documentation, accessed November 6, 2025, <https://petsc.org/release/manualpages/TS/TSSUNDIALS/>
95. Yes, that's mostly right (instead of PETSc put FATODE since PETSc TS Adjoint was... | Hacker News, accessed November 6, 2025, <https://news.ycombinator.com/item?id=28955550>
96. SUNDIALS and hypre: Exascale-Capable Libraries for Adaptive Time-Stepping and Scalable Solvers, accessed November 6, 2025, <https://www.exascaleproject.org/highlight/sundials-and-hypre-exascale-capable-libraries-for-adaptive-time-stepping-and-scalable-solvers/>
97. 1.9. Features for GPU Accelerated Computing - SUNDIALS documentation - Read the Docs, accessed November 6, 2025, <https://sundials.readthedocs.io/en/latest/sundials/GPU_link.html>
98. Using SUNDIALS on GPU-based HPC platforms - | Computing - Lawrence Livermore National Laboratory, accessed November 6, 2025, <https://computing.llnl.gov/sites/default/files/2024-11/CAC_Tutorial-SUNDIALS_HPC-2024.pdf>
99. 14. SUNDIALS Installation Procedure, accessed November 6, 2025, <https://sundials.readthedocs.io/en/v6.5.1/Install_link.html>
100. 1.1. Installing SUNDIALS, accessed November 6, 2025, <https://sundials.readthedocs.io/en/develop/sundials/Install_link.html>
101. Libceed - Consortium for the Advancement of Scientific Software, accessed November 6, 2025, <https://cass.community/software/libceed.html>
102. CEED/libCEED: CEED Library: Code for Efficient Extensible Discretizations - GitHub, accessed November 6, 2025, <https://github.com/CEED/libCEED>
103. libCEED - Exascale Computing Project, accessed November 6, 2025, <https://ceed.exascaleproject.org/libceed/>
104. libCEED - Lightweight High-Order Finite Elements Library with Performance Portability and Extensibility - SC19, accessed November 6, 2025, <https://sc19.supercomputing.org/proceedings/tech_poster/poster_files/rpost252s2-file2.pdf>
105. Interface Concepts — libCEED 0.12.0 documentation, accessed November 6, 2025, <https://libceed.org/en/latest/libCEEDapi/>
106. Introduction — libCEED 0.12.0 documentation, accessed November 6, 2025, <https://libceed.org/en/latest/intro/>
107. Getting Started — libCEED 0.12.0 documentation, accessed November 6, 2025, <https://libceed.org/en/latest/gettingstarted/>
108. Introduction — FFTX documentation - GitHub Pages, accessed November 6, 2025, <https://spiral-software.github.io/fftx/introduction.html>
109. FFT and Solver Libraries for Exascale: FFTX and SpectralPack, accessed November 6, 2025, <http://www.spiral.net/doc/pdf/fftx-ecp-poster2020_final.pdf>
110. MAGMA - NVIDIA Developer, accessed November 6, 2025, <https://developer.nvidia.com/magma>
111. MAGMA - Research Computing Documentation - University of South Florida, accessed November 6, 2025, <https://wiki.rc.usf.edu/index.php/MAGMA>
112. Dense Linear Algebra Solvers for Multicore with GPU Accelerators - IPDPS, accessed November 6, 2025, <http://www.ipdps.org/ipdps2010/ipdps2010-slides/HIPS/dense-la-IPDPS10.pdf>
113. What are the key differences between cuBLAS and other linear algebra libraries like cuSPARSE and MAGMA? - Massed Compute, accessed November 6, 2025, [https://massedcompute.com/faq-answers/?question=What%20are%20the%20key%20differences%20between%20cuBLAS%20and%20other%20linear%20algebra%20libraries%20like%20cuSPARSE%20and%20MAGMA?](https://massedcompute.com/faq-answers/?question=What+are+the+key+differences+between+cuBLAS+and+other+linear+algebra+libraries+like+cuSPARSE+and+MAGMA?)
114. New to Magma - scheduling on CPU + GPU, accessed November 6, 2025, <https://groups.google.com/a/icl.utk.edu/g/magma-user/c/kVIwnkaoCSw>
115. PyMAGMA: A Python Interface for MAGMA - TRACE: Tennessee Research and Creative Exchange, accessed November 6, 2025, <https://trace.tennessee.edu/cgi/viewcontent.cgi?article=1004&context=utk-comppubs>
116. MAGMA Users' Guide, accessed November 6, 2025, <https://icl.utk.edu/projectsfiles/magma/doxygen/>
117. carloholly/pymagma: A Python interface to the MAGMA libraries - GitHub, accessed November 6, 2025, <https://github.com/carloholly/pymagma>
118. "PyMAGMA: A Python Interface for MAGMA" by Julian Halloy, Stephen Qiu et al., accessed November 6, 2025, <https://trace.tennessee.edu/utk-comppubs/6/>
119. PyMAGMA: A Python Library for MAGMA - National Institute for Computational Sciences, accessed November 6, 2025, <https://nics.utk.edu/wp-content/uploads/sites/46/2023/02/Magma-to-Python-FPO.pdf>
120. Trilinos/Kokkos: Shared-memory programming interface and computational kernels, accessed November 6, 2025, <https://www.docs.trilinos.org/r12.16/packages/kokkos/doc/html/>
121. Trilinos/Kokkos-based strategy towards achieving a performance portable land-ice model, accessed November 6, 2025, <https://www.sandia.gov/app/uploads/sites/127/2021/11/tezaur_birs_workshop_glaciology_jan_2020_final.pdf>
122. Kokkos documentation, accessed November 6, 2025, <https://kokkos.org/kokkos-core-wiki/>
123. The Kokkos C++ Performance Portability EcoSystem, accessed November 6, 2025, <https://oaciss.uoregon.edu/E4S-Forum19/talks/Trott-E4S.pdf>
124. Kokkos: C++ Performance Portability for Production, accessed November 6, 2025, <https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2019/08/ATPESC_2019_Track-2_3_8-1_830am_Trott-Kokkos.pdf>
125. Toward performance-portable PETSc for GPU-based exascale systems - OSTI, accessed November 6, 2025, <https://www.osti.gov/pages/biblio/1834595>
126. [2011.00715] Toward Performance-Portable PETSc for GPU-based Exascale Systems, accessed November 6, 2025, <https://arxiv.org/abs/2011.00715>
127. PETSc: Features: GPU support, accessed November 6, 2025, <https://web.cels.anl.gov/projects/petsc/vault/OLD/features/gpus.html>
128. GPU Support Roadmap — PETSc v3.24.0-158-gdafee7125fca documentation, accessed November 6, 2025, <https://petsc.org/main/overview/gpu_roadmap/>
129. Toward Performance-Portable PETSc for GPU-based Exascale Systems - eScholarship, accessed November 6, 2025, <https://escholarship.org/content/qt9wg9p5kj/qt9wg9p5kj.pdf>
130. Porting the Ginkgo Package to AMD's HIP Ecosystem - Better Scientific Software (BSSw), accessed November 6, 2025, <https://bssw.io/items/porting-the-ginkgo-package-to-amd-s-hip-ecosystem>
131. Overview of the Ginkgo library design using the backend model for... - ResearchGate, accessed November 6, 2025, <https://www.researchgate.net/figure/Overview-of-the-Ginkgo-library-design-using-the-backend-model-for-platform-portability_fig1_379746526>
132. Requirements · hypre-space/hypre Wiki - GitHub, accessed November 6, 2025, <https://github.com/hypre-space/hypre/wiki/Requirements>
133. General Information — hypre 3.0.0 documentation - Read the Docs, accessed November 6, 2025, <https://hypre.readthedocs.io/en/latest/ch-misc.html>
134. Porting Sparse Linear Algebra to Intel GPUs, accessed November 6, 2025, <https://publikationen.bibliothek.kit.edu/1000148428/149841836>
135. Preparing Ginkgo for AMD GPUs – A Testimonial on Porting CUDA Code to HIP, accessed November 6, 2025, <https://publikationen.bibliothek.kit.edu/1000131542/140334625>
136. Trilinos Support on AMD and Intel GPUs: SAKE Project, accessed November 6, 2025, <https://trilinos.github.io/pages/community/trilinos_user_meetings/trilinos_user-developer_group_meeting_2021/2021-11-30/06.2-Kelly-TUG_2021_Sake_solvers_revised.pdf>
137. GPU Support - STRUMPACK v7.2.0, accessed November 6, 2025, <https://portal.nersc.gov/project/sparse/strumpack/master/GPU_Support.html>
138. A GPU accelerated sparse direct solver and preconditioner with block low rank compression - eScholarship, accessed November 6, 2025, <https://escholarship.org/content/qt7tn9n67r/qt7tn9n67r_noSplash_3c7c1d08b881a9980fe7328cd40938e5.pdf>
139. Investigating the HIP programming model with regards to portability and performance portability, accessed November 6, 2025, <https://events.gwdg.de/event/243/contributions/506/attachments/142/179/FINAL_Seminar_on_Performance_Portable_Programming_of_HPC_Applications_Topic_HiP.pdf>
140. Porting NVIDIA CUDA code to HIP - AMD ROCm documentation, accessed November 6, 2025, <https://rocm.docs.amd.com/projects/HIP/en/docs-develop/how-to/hip_porting_guide.html>
141. HIP Porting Guide - AMD ROCm documentation, accessed November 6, 2025, <https://rocm.docs.amd.com/projects/HIP/en/docs-5.7.1/user_guide/hip_porting_guide.html>
142. Why does not someone create a startup specializing in sycl/ROCm that runs on all types of GPUs - Reddit, accessed November 6, 2025, <https://www.reddit.com/r/ROCm/comments/1hf8vcv/why_does_not_someone_create_a_startup/>
143. Releases · pghysels/STRUMPACK - GitHub, accessed November 6, 2025, <https://github.com/pghysels/STRUMPACK/releases>
144. HYPRE: Scalable Linear Solvers and Multigrid Methods - | Computing - Lawrence Livermore National Laboratory, accessed November 6, 2025, <https://computing.llnl.gov/projects/hypre-scalable-linear-solvers-multigrid-methods>