

EXANLA: HORIZON EUROPE PROPOSAL STRATEGY

Stable Runtime-Oriented Dense Numerical Infrastructure for Exascale Applications

JLESC Workshop, May 2026 | Xinzhe Wu, Edoardo Di Napoli | Jülich Supercomputing Centre

EXANLA STRATEGY OVERVIEW

Horizon Europe INFRA-TECH Vision Core Architecture

- **Application-driven exascale dense numerical infrastructure:** Custom targeted stacks modeled directly around production workflow profiles.
- **ABI-stable runtime-oriented software stack:** Standardizing binary communication pathways between high-level apps and low-level drivers.
- **Workflow-native numerical methods:** Algorithmic frameworks executing gracefully across complex operational cycles.
- **Long-lived scientific software sustainability:** Guaranteeing decoupling mechanisms to shield applications from platform shifts.

Central Objective

Building persistent, robust, and evolution-resilient numerical baselines for European RI upgrades.

JÜLICH Forschungszentrum

THE REAL PROBLEM: EXASCALE SOFTWARE CRISIS

Disconnection Between Evolving Accelerators and Multi-Decade Codebases

Hardware Reality

- **Heterogeneous systems operational:** *JUPITER, LUMI, LEONARDO, Alps*.
- Massive accelerator raw computing throughput.
- Rapid vendor backend evolution cycles.
- Frequent ABI / runtime environment instability.

Scientific Software Reality

- **Long-lived applications:** *QE, CP2K, SIRIUS, BigDFT, Yambo, DFTB+*.
- Severe operational fragmentation issues:
  - Layout mismatch & redistribution overhead.
  - Compile-time rigidity & backend failures.
  - Extreme recompilation and tuning burdens.

Core Structural Statement

The primary bottleneck is **no longer isolated kernel FLOPS**. The strategic limits are now defined by **integration survivability, runtime interoperability, workflow-native execution,** and **stable multi-platform deployments**.

EXANLA POSITIONING: WHAT IT IS AND IS NOT

Defining the Software Waist Decoupling Layer

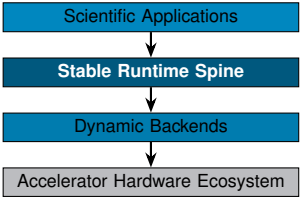
ExaNLA is NOT:

- ✗ Another basic BLAS replacement package.
- ✗ Another collection of hand-tuned standalone kernels.
- ✗ Another isolated algebraic eigensolver wrapper.
- ✗ Another passive, high-overhead portability layer.

ExaNLA IS:

- ✓ A **stable, runtime-oriented numerical infrastructure layer** for production exascale scientific workflows.

Central Vision: Narrow Software Waist



Strategic Shift: "Kernel-centric optimization" ⟶ **"Workflow-centric infrastructure"**

PROPOSAL PHILOSOPHY: INFRASTRUCTURE-CENTERED

Moving Beyond Isolated Mathematical Subroutine Optimizations

Traditional HPC Subroutine Approach

- Isolated optimization of `GEMM`, `QR`, `TRSM`.
- Target max peak performance metrics in a vacuum.
- Rigorous static compile-time specializations.

ExaNLA Systemic Infrastructure Approach

- Preserve strict data locality across workflows.
- Drastically minimize memory redistribution stages.
- Stabilize runtime binaries; guarantee long-term compatibility.

Key Algorithmic Metamorphosis

Numerical algorithms are **not isolated deliverables**. They are systematically recast to be:

Runtime-aware · *Workflow-native* · *Topology-aware* · *Descriptor-driven*

CORE SCIENTIFIC HYPOTHESIS

The Foundations for Resilient Exascale Deployments

The ExaNLA Algorithmic Hypothesis

Exascale dense numerical software architectures mandate a radical pivot toward:

1. **Stable runtime execution semantics**.
2. **Persistent layout-aware, in-place execution blocks**.
3. **Runtime-adaptive, variable precision mathematical pipelines**.
4. **Continuous verification and empirical validation schemes**.

rather than relying on isolated static kernels, compile-time-only abstractions, and fragile backend coupling ties.

Long-term Goal: Enable applications to natively survive continuous multi-vendor architectural transitions **without manual code rewrites**.

PROPOSAL STRUCTURE: THREE FOUNDATIONAL PILLARS

Cohesive Task Grouping Mapping Key Capabilities

Pillar A
Stable Runtime Spine

- Frozen binary C ABI interface
- Data containment via opaque handles
- Dynamic hardware dispatch routing
- Structural execution descriptors
- Multi-language interface bindings

Pillar B
Workflow Pipelines

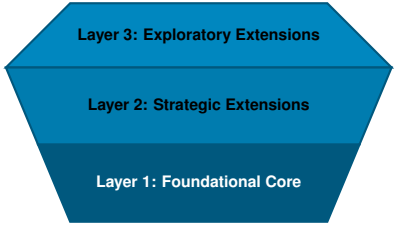
- Layout-aware memory executions
- Data redistribution minimization
- Communication-avoiding reductions
- Adaptive runtime mixed-precision execution
- Persistent GPU-resident pipelines

Pillar C
Continuous Validation

- Scientific database: **MatrixHub**
- Mini-app driven validation layer
- Continuous integration via **exaCB**
- Benchmarking profiles with **JUBE**
- Empirical validation setups
- Mixed-precision safety thresholds

LAYERED TASK MODEL

Structural Prioritization Minimizing Feasibility Risk



Layer 1 — Foundational Core (Mandatory):

- Establishes MatrixHub, ABI spine, dispatch layers, language bindings, and basic layout reshaping.

Layer 2 — Strategic Extensions (Value-Add):

- Energy telemetry, topology performance characterization profiles.

Layer 3 — Exploratory Extensions (Future-Proof):

- AI4S operator extensions, resilience execution hooks.

Key Guideline: Only Layer 1 defines the mandatory proposal deliverables core.

WP1 — CONTINUOUS SCIENTIFIC VALIDATION INFRASTRUCTURE

Quantifying Performance, Portability, and Energy Across Platforms

Objective: Build the continuous empirical testing backbone for European exascale numerical lines.

Task 1.1 — Continuous Benchmarking

- **exaCB + JUBE Integration:** Automated execution pipelines across diverse hardware architectures.
- Regression tracking under strict compute budgets.

Task 1.2 — MatrixHub Infrastructure

- European repository for real scientific matrices.
- Categorized metadata mapped directly to production application failures.

Task 1.3 — Energy Telemetry

✗ NOT: Autonomous hardware scheduling.

✓ **BUT:** Non-intrusive energy profiling and software-level execution hints.

Task 1.4 — Topology Cliff Diagnosis

✗ NOT: Black-box AI optimization rules.

✓ **BUT:** Analytical identification of memory redistribution hotspots and NUMA bottlenecks.

BEYOND MATRICES: MINI-APP DRIVEN VALIDATION LAYER

Aligning ExaNLA with MaX-style full workflow methodology

Key Shift in Methodology

ExaNLA is not validated on isolated matrices alone, but on **end-to-end scientific mini-app workflows**.

Why Mini-Apps?

- Capture full solver pipelines (not just GEMM / QR)
- Include data movement + redistribution + I/O effects
- Represent real application bottlenecks (DFT, GW, BSE, GNN)
- Enable reproducible cross-stack benchmarking

Mini-App Examples

- Electronic structure: CP2K / Quantum ESPRESSO reduced workflow
- Excited states: Yambo / BSE pipeline kernels
- Sparse-dense hybrid solvers (SLEPc-like workflows)
- AI4S kernels: attention + dense linear algebra coupling

Role in ExaNLA Stack

Mini-apps serve as **system-level correctness and performance validators** bridging: Kernel → Library → Runtime Spine → Hardware

EXANLA BENCHMARK ECOSYSTEM

From MatrixHub to Full Scientific Workflow Validation

Three-Tier Benchmark Strategy

- **Tier 1: MatrixHub**
  - Raw dense matrices (WG1 baseline)
  - Performance cliffs + scaling studies
- **Tier 2: Micro-Kernels**
  - GEMM / QR / GEP / Cholesky variants
  - Precision & layout stress tests
- **Tier 3: Mini-Apps (NEW)**
  - End-to-end workflows
  - Runtime + communication + I/O effects
  - Real solver behavior under system load

...

Why This Matters for EU INFRA

- Matches MaX ecosystem validation philosophy
- Enables cross-code reproducibility (CP2K QE SIRIUS)
- Provides procurement-grade benchmarking evidence
- Directly supports Horizon Europe “application-driven” requirement

Outcome

A unified benchmark stack spanning **kernel** $\rightarrow$ **workflow** $\rightarrow$ **application**.

MINI-APP INTEGRATION INTO WORKING GROUPS

Closing the Loop Between WG1–WG3 and Applications

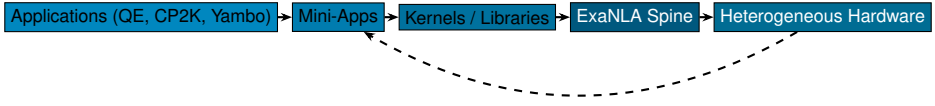
WG	Current Focus	Mini-App Extension
WG1	Matrix performance + scaling cliffs	Mini-app execution traces (runtime + memory + network coupling)
WG2	Functionality survey across applications	Mini-app construction from real application slices
WG3	ABI + runtime spine design	Mini-app-driven validation of dispatch + routing decisions

Key Structural Insight

Mini-apps act as the **missing intermediate abstraction layer between applications and kernels**.

END-TO-END VALIDATION LOOP

From Scientific Applications to Hardware Feedback



Key Principle

Validation is **closed-loop**: hardware feedback continuously refines mini-apps, kernels, and runtime decisions.

WP2 — APPLICATION-DRIVEN NUMERICAL WORKFLOW DESIGN

Extracting Structural Constraints and Building Layout Pipelines

Objective: Extract strict computational dependencies from real applications to drive software optimization.

Task 2.1 — Requirements Observatory

- Continuous code surveys covering **22+ flagship applications**.
- Tracking data structure shapes, communication steps, and memory constraints.

Task 2.2 — Precision Envelope

- Identifying numerical safety boundaries when scaling from FP64 down to lower precisions.

Task 2.3 — Emerging Operator Classes

✗ NOT: Generic AI model training frameworks.

✓ **BUT:** Dense attention contractions and graph-induced dense kernels in AI4S workflows.

Task 2.4 — Persistent Layout Pipelines

- **Core Demonstrator Block:** Asynchronous out-of-band matrix reshaping to ensure data stays resident inside GPU memory.

WP3 — STABLE RUNTIME SPINE AND NUMERICAL ORCHESTRATION

Authoring the Execution Middleware of the ExaNLA Ecosystem

Objective: Construct the foundational binary waist and dynamic execution routers.

Task 3.1 — Stable C ABI Spine

- Frozen binary interfaces with opaque handles.
- Decouples core libraries; updates backends transparently without app recompilations.

Task 3.2 — Language Interoperability

- One-time binding model targeting Python C-FFI, Julia, and Fortran `iso_c_binding`.

Task 3.3 — Runtime Orchestration Routing

✗ NOT: Brittle, unpredictable AI engine schedulers.

✓ **BUT:** Topology-aware backend routing based on runtime descriptor evaluation.

Task 3.4 — Resilience Semantics

✗ NOT: Heavy system-level transparent MPI recovery.

✓ **BUT:** Light, handle-driven failure hooks protecting execution workflows.

THE ALGORITHMIC STRATEGY: WHY WE INNOVATE NUMERICAL METHODS

Reframing Mathematical Inventions Within Runtime Frameworks

Algorithmic Clarification

ExaNLA is **not a passive software wrapper**. Mathematical algorithmic innovations remain the core scientific engine driving our project.

Traditional Mathematical Outputs

- ✗ Isolated, custom, single-kernel code paths.
- ✗ Maximizing peak FLOPS results for narrow papers.
- ✗ Static mathematical assumptions ignored by applications.

ExaNLA Algorithmic Outputs

- ✓ Runtime-integrated numerical blocks.
- ✓ Topology-aware distributed communication bypasses.
- ✓ Descriptors that dynamically drive precision switching.

FOUR RULES FOR EXANLA ALGORITHMIC DESIGN

Mandatory Directives for Mathematicians and Library Authors

Rule 1: Application-Proven

Algorithms must target real problems exposed in production codes, verified via MatrixHub entries.

Rule 2: Runtime-Integrated

Algorithms must communicate via dynamic execution handles, data descriptors, and online parameters.

Rule 3: Workflow-Measurable

Metrics of success track the reduction of end-to-end simulation time, data movement, and energy footprints.

Rule 4: Infrastructure-Enabling

Algorithmic implementations must feed architectural metadata back to the runtime routing engines.

RECOMMENDED NUMERICAL INNOVATION AREAS

Actionable Domains of Algorithmic Upgrades

A. Runtime-Adaptive Multi-Precision

- Iterative refinement routines utilizing hardware Tensor Cores dynamically.
- Active residual monitoring triggering runtime precision alterations.

B. Persistent Layout Numerical Flows

- Out-of-band asynchronous memory layout modifications.
- Matrix algorithms preserving block data structures across solver steps.

C. Communication-Avoiding Reductions

- Distributed `CA-QR` and `CA-Cholesky` implementations matched to non-uniform network configurations.

D. Routable Generalized Solvers

- Adaptive eigensolver pipelines (e.g., polynomial filters) switching execution steps based on matrix spectrum parameters.

FLAGSHIP DEMONSTRATOR: INTEGRATED EXECUTION FLOW

Validating the Infrastructure via an End-to-End Scientific Showcase Pipeline

Target Case Study: Generalized Hermitian Eigenproblem Workflow (*QE, SIRIUS, CP2K, Yambo*)



Quantifiable Validation Metrics

- Total avoidance of temporary CPU-GPU memory copies.
- Maintaining data completely resident inside GPU memory spaces.
- Robust cross-vendor platform compatibility verifications.
- Complete minimization of execution bottlenecks.

WHAT MAKES EXANLA DIFFERENT

Paradigm Shift Tracking Essential Structural Criteria

Dimension	Traditional Mathematical Libraries	ExaNLA Paradigm Stack
Design Core	Kernel-centric optimization (isolated functions)	Workflow-native application integration
Interfaces	Rigorous, static compile-time headers	Dynamic runtime descriptor abstractions
Backend Coupling	Hardcoded ties forcing total recompilations	Decoupled ABI-stable narrow waist
Memory Strategy	Rigid layout criteria causing data copies	Out-of-band asynchronous data transpositions

Core Paradigm Shift: ExaNLA does not merely optimize numerical kernels.

ExaNLA stabilizes the core dense numerical infrastructure layer for European exascale science.

JÜLICH Forschungszentrum

FINAL STRATEGIC POSITIONING

A Resilient European Infrastructure Framework for Sustainable Discovery

Key Infrastructure Commitments

- **Stable Software Deployments:** Zero application-level rewrites or re-compilations during platform upgrades.
- **Runtime-Oriented Adaptability:** Dynamically scales capabilities to match hardware properties online.
- **Accelerator-Native Residency:** Minimizes high-cost internode data overheads via continuous execution tracking.
- **Long-Term Software Sustainability:** Protecting multi-million-euro investments into flagship European codes.

Concluding Vision

The central challenge of exascale computing is no longer isolated kernel execution performance. The structural challenge is building sustainable runtime infrastructure for decades-long scientific workflows operating on rapidly evolving accelerator systems.