RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

FRANCE 2030
PROGRAMME DE RECHERCHE
NUMÉRIQUE POUR L'EXASCALE

High-Performance Spectral Element Operators on GPUs

A GEMM-Based Reformulation for Modern Accelerators

Alexandre Roget

February 23, 2026

NumPEx Project - 2026 Annual Meeting of Exa-DI

Efficient High-order Discretisation for PDEs at Exascale

WG1 Objectives:

- Enable efficient high-order PDE discretisations on exascale hardware.
- Promote co-design and co-development through well-specified proxy-apps and mini-apps.
- Explore performance portability and architecture-dependent trade-offs.

WG1 is driven by two application demonstrator groups: `FUnTiDES` and `Feel++`.

FUnTiDES: Fast Unstructured Time Dynamic Equation Solver

`FUnTiDES` is a suite of lightweight proxy applications representing real scientific HPC workloads.

- Designed as a performance benchmark for comparing HPC systems and programming models.
- Includes two numerical solvers for the second-order acoustic wave equation:
  - SEM — Spectral Element Method (Galerkin FEM)
  - FD — Finite Differences (stencil-based)
- Portable across multiple architectures (CPU, GPU).

Proxy-Kernels

`Proxy-Kernels` is a standalone library extracted from the `FUnTiDES` codebase as a standalone component.

- Focused specifically on the discretization layer of the SEM.
- Designed as an experimental playground to compare alternative SEM formulations.

Global SEM system

The semi-discrete Spectral Element formulation reads:

$$\mathbf{M}\,\frac{d^2\mathbf{p}}{dt^2} + \mathbf{K}\,\mathbf{u} = \mathbf{f}.$$

- GLL collocation $\Rightarrow$ diagonal mass matrix $\mathbf{M}$ $\Rightarrow$ explicit time integration.
- Dominant cost: repeated evaluation of the stiffness operator $\mathbf{Ku}$.

Classical Quadrature-Based Formulation

The local stiffness operator in a classical SEM implementation reads:

$$\mathbf{K} = c^2 \sum_{p,q,r} w_p w_q w_r |J_{pqr}| \sum_{\alpha,\beta=1}^{3} G_{pqr}^{\alpha\beta} \left(D^\alpha \otimes D^\beta \right)$$

- Evaluation at each quadrature point (p, q, r)
- Directional derivatives via D^α
- Metric tensor $G^{\alpha\beta}$ applied pointwise

Why Rethink the Classical Formulation?

Modern GPUs achieve peak performance when:

- Computations exhibit high arithmetic intensity
- Data access is regular and structured
- Workloads are expressed as large dense linear algebra kernels

However, the classical quadrature-based implementation:

- Relies on deeply nested element-wise loops
- Involves many small tensor contractions
- Offers limited data reuse

Tensorial Reformulation (Matrix-Free)

Applying the local stiffness operator to u can be written as:

$$(Ku)_{ijk} = c^2 \sum_{\alpha=1}^{3} \sum_{\beta=1}^{3} \left[D_\alpha^\top \left(\mathbf{W} \odot |J| \odot G^{\alpha\beta} \odot (D_\beta u) \right) \right]_{ijk}$$

where the metric-weighted tensor

$$\widetilde{G}^{\alpha\beta} = \mathbf{W} \odot |J| \odot G^{\alpha\beta}$$

is precomputed at quadrature points.

- $(D_\beta u)_{ijk}$: one-dimensional directional derivative operator in direction β
- Pointwise application of geometric coefficients
- Backward contraction with $D_\alpha^\top$

Tensorial GEMM-based Workflow — Precomputation

For a polynomial degree N, let $n = N + 1$.

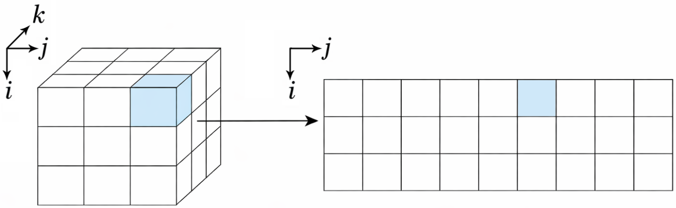
1. Initialization of the one-dimensional reference gradient operator $\mathbf{D} \in \mathbb{R}^{n \times n}$
   - rows $\rightarrow$ Gauss–Lobatto nodes
   - columns $\rightarrow$ polynomial basis indices
2. Precompute the metric-weighted tensor $\widetilde{G}^{\alpha\beta}$ at all quadrature points.

Tensorial GEMM-based Workflow — Local Operator Application

1. Reshape the local nodal field $\mathbf{u} \in \mathbb{R}^{n \times n \times n}$ into directional matrices $U_\xi, U_\eta, U_\zeta \in \mathbb{R}^{n \times n^2}$
2. Compute reference-space directional derivatives
   $\rightarrow$ 3 GEMM operations $\mathbb{R}^{n \times n} \times \mathbb{R}^{n \times n^2}$
3. Application of the precomputed weighted metric tensor $\widetilde{G}^{\alpha\beta}$ pointwise
4. Backward contractions with $\mathbf{D}^\top$
   $\rightarrow$ 3 GEMM operations $\mathbb{R}^{n \times n} \times \mathbb{R}^{n \times n^2}$
5. Inverse reshaping
   - restore the original (i, j, k) nodal indexing
   - sum directional contributions to recover $(K\mathbf{u})_{ijk}$

Stiffness Operator Computation (1/5)

- **Reshape the local nodal field into directional matrices**

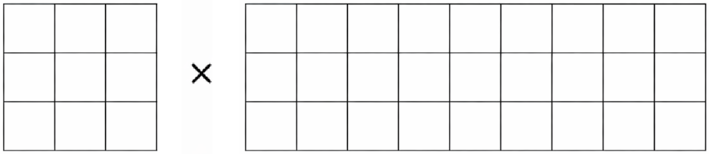


$u \in \mathbb{R}^{n\times n\times n}$ $\quad$ $U_\xi \in \mathbb{R}^{n\times n^2}$

$$\boxed{\begin{aligned} U_\xi(i,\ j+k\,n) &= u(i,j,k)\\ U_\eta(j,\ i+k\,n) &= u(i,j,k)\\ U_\zeta(k,\ i+j\,n) &= u(i,j,k)\end{aligned}}$$

Stiffness Operator Computation (2/5)

- **Compute reference-space directional derivatives**

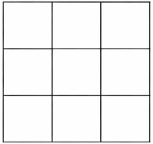


$D \in \mathbb{R}^{n \times n}$ $\quad$ $U_\xi \in \mathbb{R}^{n \times n^2}$

$$\boxed{\begin{aligned} \partial_\xi u &= D\,U_\xi \\ \partial_\eta u &= D\,U_\eta \\ \partial_\zeta u &= D\,U_\zeta \end{aligned}}$$

Stiffness Operator Computation (3/5)

- **Application of the precomputed weighted metric tensor**



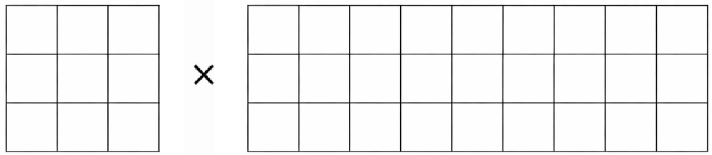
$\widetilde{G}^{\alpha\beta} \in \mathbb{R}^{3\times 3}$

$$\boxed{\widetilde{G}^{\alpha\beta} = \mathbf{W} \odot |J| \odot G^{\alpha\beta}}$$

$$\boxed{\begin{aligned} F_\xi &= \widetilde{G}^{11}\,\partial_\xi u + \widetilde{G}^{12}\,\partial_\eta u + \widetilde{G}^{13}\,\partial_\zeta u \\ F_\eta &= \widetilde{G}^{21}\,\partial_\xi u + \widetilde{G}^{22}\,\partial_\eta u + \widetilde{G}^{23}\,\partial_\zeta u \\ F_\zeta &= \widetilde{G}^{31}\,\partial_\xi u + \widetilde{G}^{32}\,\partial_\eta u + \widetilde{G}^{33}\,\partial_\zeta u \end{aligned}}$$

Stiffness Operator Computation (4/5)

▷ **Backward contractions with $\mathbf{D}^\top$**

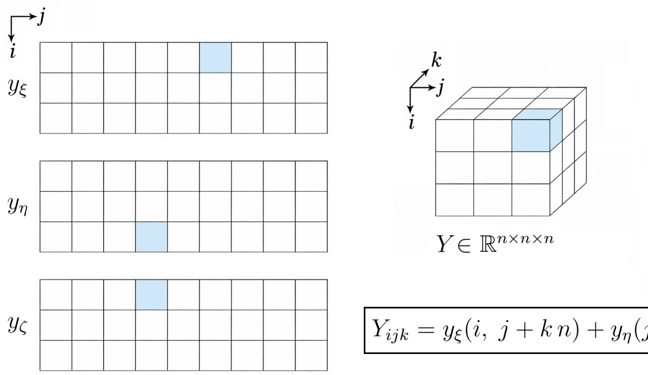


$D^\top \in \mathbb{R}^{n\times n}$ $\quad$ $F_\xi \in \mathbb{R}^{n\times n^2}$

$$\boxed{\begin{aligned} y_\xi &= D^\top F_\xi \\ y_\eta &= D^\top F_\eta \\ y_\zeta &= D^\top F_\zeta \end{aligned}}$$

Stiffness Operator Computation (5/5)

▷ **Inverse reshaping**



$$\boxed{Y_{ijk} = y_\xi(i,\ j + k\,n) + y_\eta(j,\ i + k\,n) + y_\zeta(k,\ i + j\,n)}$$

GEMM-based Implementation

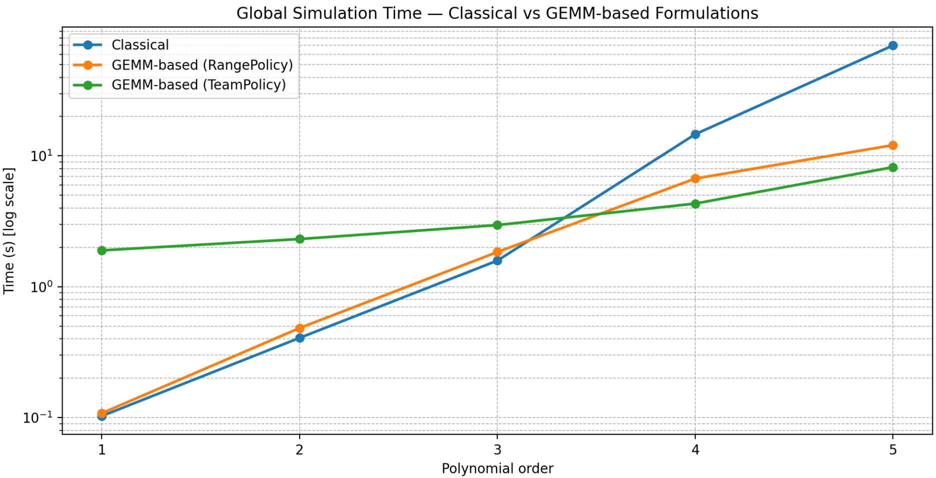
Method 1 — One element per thread (`Kokkos::RangePolicy`)

- Directional tensors in local (stack) memory
- Sequential execution of 6 GEMMs
- No intra-element parallelism

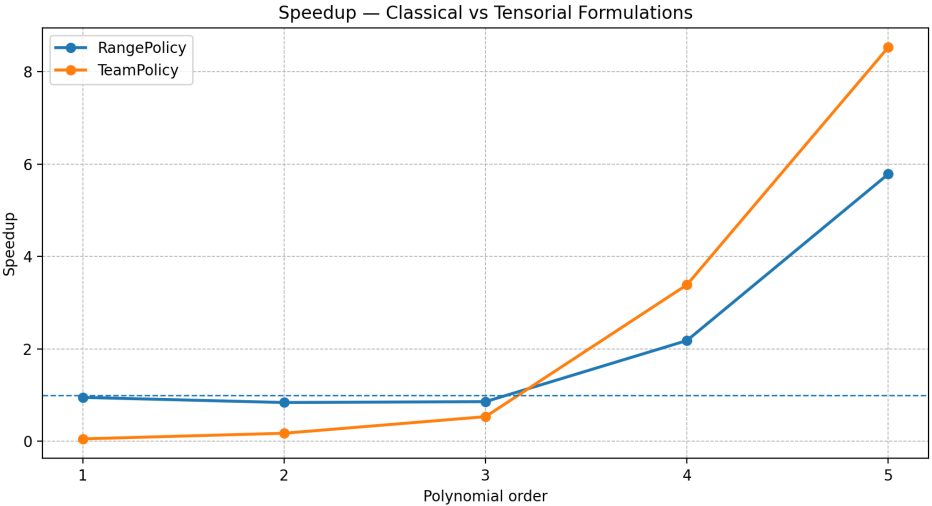
Method 2 — One element = one team of threads (`Kokkos::TeamPolicy`)

- Shared scratch memory for directional tensors
- Cooperative GEMM execution (`KokkosBatched::TeamVectorGemm`)
- `team_barrier()` between pipeline stages

Performance Impact of the GEMM-Based Reformulation (1/2)



Performance Impact of the GEMM-Based Reformulation (2/2)



Next Steps

1. **Multi-element Batched GEMM**
   - Aggregate multiple elements to increase GEMM sizes
   - Improve arithmetic intensity and GPU occupancy
2. **Tensor Core Exploitation (CUDA WMMA)**
   - Reformulate contractions for WMMA fragments
   - Mixed-precision opportunities
3. **Higher Polynomial Orders**
   - Current limitation: $N \leq 5$
   - Larger tensors $\Rightarrow$ better GPU utilization

Conclusion

- High-order SEM operators naturally expose a tensor-product structure.
- Reformulating the stiffness operator into dense tensor contractions enables a GEMM-based implementation well suited to GPUs.
- Significant speedups observed on GPUs, even without Tensor Core optimizations.
- Further gains are expected through batching, improved intra-element parallelism, and WMMA-based kernels.

Questions?