A Matrix-Free High-Dimensional DG Approach for Mitigating the Rays-Effect in Phase-Space Advection.

MFEM community workshop 2024 – Livermore, USA

Yohann Dudouit Veselin Dobrev Terry Haut Milan Holec John Loffeld Jan Nikl Lee Ricketson Amit Rotem

October 22nd–24th 2024

LLNL-PRES-870722

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

The Rays-Effect Problem

Model Problem:

$$\vec{\Omega} \cdot \nabla f(X, \vec{\Omega}, E) = S(X, \vec{\Omega}, E)$$

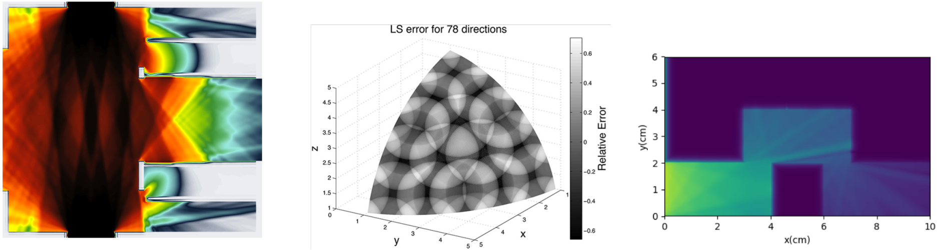


Figure: Snapshots showcasing the rays-effect in different numerical simulations.

The Strategy: The Generalized S_N methodology (GSN)

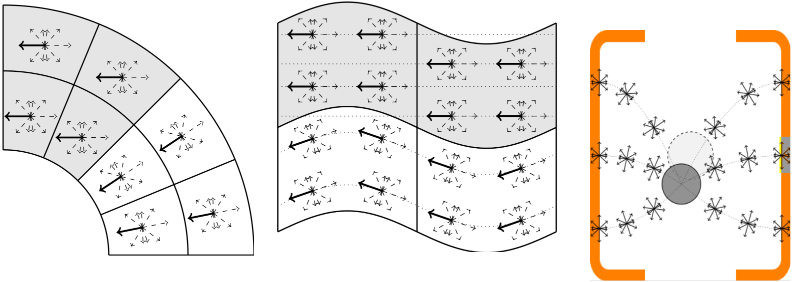
(Milan Holec)

General Coordinate System

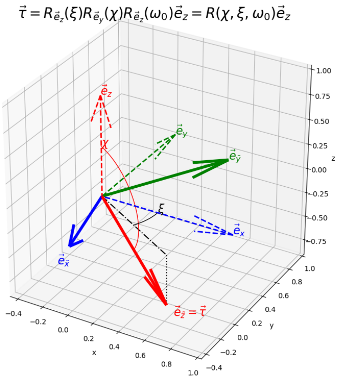
Derive general change of coordinates at the continuous level aiming at mitigating ray effects at the discrete level.

Multi-Dimensional DG Discretization

Develop a novel high-order matrix-free multi-dimensional DG method.



General Change of Coordinates for the Momentum Mesh



Rotation of the Momentum Mesh:

$\forall X \in K,\ \forall \omega \in [0, 2\pi],\ \forall \mu \in [-1, 1]$

$$\vec{\Omega}(X, \omega, \mu, e) = R(\vec{\zeta}(X), \omega_0(X)) \begin{bmatrix} cos(\omega)\sqrt{1-\mu^2} \\ sin(\omega)\sqrt{1-\mu^2} \\ \mu \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

- Analytical $\vec{\zeta}(X) \Rightarrow$ Analytical-GSN,
- Numerical $\vec{\zeta}(X) \Rightarrow$ Flux-GSN.

Derivation of the Rotation Matrix

$\nabla f = J^{-T}\tilde{\nabla} f$, and assuming that $\frac{\partial \mathbf{q}}{\partial \tilde{\mathbf{p}}} = 0$, we get

$$J^{-1} = \begin{pmatrix} \frac{\partial \mathbf{q}}{\partial \tilde{\mathbf{q}}} & \frac{\partial \mathbf{q}}{\partial \tilde{\mathbf{p}}} \\ \frac{\partial \mathbf{p}}{\partial \tilde{\mathbf{q}}} & \frac{\partial \mathbf{p}}{\partial \tilde{\mathbf{p}}} \end{pmatrix}^{-1} = \begin{pmatrix} \frac{\partial \mathbf{q}}{\partial \tilde{\mathbf{q}}} & 0 \\ \frac{\partial \mathbf{p}}{\partial \tilde{\mathbf{q}}} & \frac{\partial \mathbf{p}}{\partial \tilde{\mathbf{p}}} \end{pmatrix}^{-1} = \begin{pmatrix} \frac{\partial \mathbf{q}}{\partial \tilde{\mathbf{q}}}^{-1} & 0 \\ -\frac{\partial \mathbf{p}}{\partial \tilde{\mathbf{p}}}^{-1} \frac{\partial \mathbf{p}}{\partial \tilde{\mathbf{q}}} \frac{\partial \mathbf{q}}{\partial \tilde{\mathbf{q}}}^{-1} & \frac{\partial \mathbf{p}}{\partial \tilde{\mathbf{p}}}^{-1} \end{pmatrix} \tag{1}$$

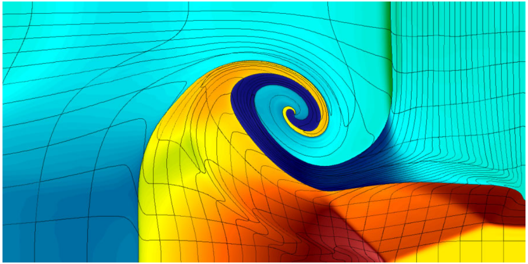
Thus,

$$\vec{\Omega}.\nabla f = \vec{\tilde{\Omega}}.J^{-T}\tilde{\nabla} f = J^{-1}\vec{\tilde{\Omega}} \cdot \tilde{\nabla} f \tag{2}$$

The mesh data-structures

Three mesh data-structures are currently supported:

- *Unstructured AMR curved* mesh: using the **mfem::Mesh** class,
- *Structured* mesh: e.g. Cartesian mesh (very computationally efficient and almost zero memory footprint),
- *Cartesian product* mesh: enable simple construction of high-dimension meshes by representing a Cartesian product mesh between either structured and unstructured meshes without any additional memory footprint.



The high-dimension mesh interface

Mesh functions:

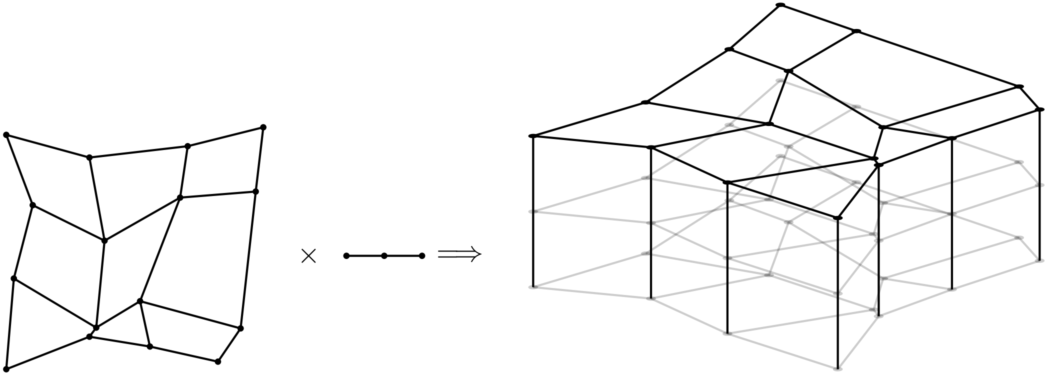
- GetCell – $K \in \tau_h$ and $\hat{F} \in \partial K$: Returns an object representing a mesh cell,
- GetFaceNeighborInfo – $\mathcal{N}_{\hat{F}}(K)$: Returns a neighboring cell info based on a reference face.

Cell functions:

- ComputePhysicalCoordinates – $\mathbf{x}_q := F(\hat{\mathbf{x}}_q)$: Compute the physical coordinates of a point in reference space,
- ComputeJacobian – $J(\hat{\mathbf{x}}_q)$: Compute the Jacobian matrix of the cell mapping at a given point in reference space,
- GetReferenceNormal – $\hat{\mathbf{n}}$: Returns the normal of a face in reference space.

$\rightarrow$ The simplicity of the mesh interface guarantees *easy support* for new mesh data-structures and mesh data-structures existing in other libraries.

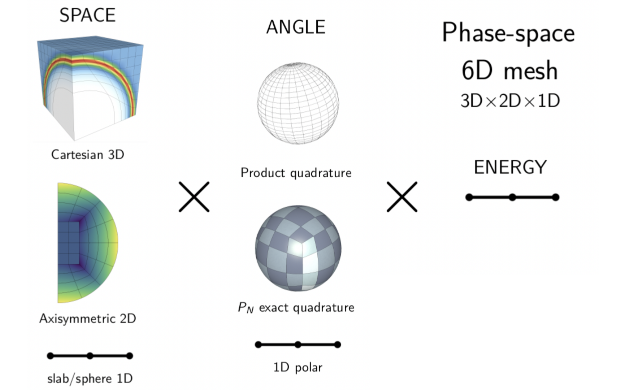
Simple Cartesian product mesh example



$\Longrightarrow$ The 3D mesh has the same memory footprint as 2D + 1D (Not 2D $\times$ 1D).

Cartesian Product Mesh for Phase Space

Tensor Product Meshes



Advantages

- Allow mixing structured and unstructured meshes,
- Low memory footprint,
- Tensor product elements by construction,
- Block diagonal Jacobians.

Discretization approach: Upwind Discontinuous Galerkin method

Find $u \in V_h$ such that:

$$\forall K \in \tau_h,\ \forall v_K \in V_K,\ \int_K u\vec{\Omega} \cdot \nabla v_K - \int_{\partial K} \vec{\Omega} \cdot \vec{n} u_{\text{upwind}} v_K = \int_K f\, v_K$$

Use matrix-free approach to mitigate the curse of dimensionality:

	Memory	Flops	Arithmetic Intensity
Sparse-Matrix	$\mathcal{O}(n(p+1)^{2d})$	$\mathcal{O}(n(p+1)^{2d})$	$\mathcal{O}(1)$
Matrix-Free	$\mathcal{O}(n(p+1)^{d})$	$\mathcal{O}(n(p+1)^{2d})$	$\mathcal{O}(p^{d})$
Matrix-Free with Sum Factorization	$\mathcal{O}(n(p+1)^{d})$	$\mathcal{O}(nd(p+1)^{d+1})$	$\mathcal{O}(dp)$

$\Rightarrow$ Up to $\underline{\mathcal{O}((p+1)^d)\text{ speedup}}$ when compared to a Sparse-matrix approach.

Matrix-Free Algorithm for Advection

Volume contribution

1. Express integral in reference coordinates,
2. Use a quadrature rule to approximate the integral by a sum,
3. Transform the sum in a sequence of operations.

Volume contributions:

$$A^K \approx \sum_q \omega_q det(J(\hat{\mathbf{x}}_q))\left(\hat{u}(\hat{\mathbf{x}}_q)\,\vec{\Omega}(\mathbf{x}_q)\right) \cdot J^{-T}(\hat{\mathbf{x}}_q)\hat{\nabla}\hat{v}(\hat{\mathbf{x}}_q) = G^T D_K B$$

where

$$B = \begin{bmatrix} \hat{\varphi}_0(\hat{\mathbf{x}}_0) & \cdots & \hat{\varphi}_N(\hat{\mathbf{x}}_0) \\ \vdots & \ddots & \vdots \\ \hat{\varphi}_0(\hat{\mathbf{x}}_Q) & \cdots & \hat{\varphi}_N(\hat{\mathbf{x}}_Q) \end{bmatrix}, \quad G = \begin{bmatrix} \hat{\nabla}\hat{\varphi}_0(\hat{\mathbf{x}}_0) & \cdots & \hat{\nabla}\hat{\varphi}_N(\hat{\mathbf{x}}_0) \\ \vdots & \ddots & \vdots \\ \hat{\nabla}\hat{\varphi}_0(\hat{\mathbf{x}}_Q) & \cdots & \hat{\nabla}\hat{\varphi}_N(\hat{\mathbf{x}}_Q) \end{bmatrix}, D_K = \begin{bmatrix} \omega_0 det(J(\hat{\mathbf{x}}_0))J^{-1}(\hat{\mathbf{x}}_0)\vec{\Omega}(\mathbf{x}_0) & & 0 \\ & \ddots & \\ 0 & & \omega_Q det(J(\hat{\mathbf{x}}_Q))J^{-1}(\hat{\mathbf{x}}_Q)\vec{\Omega}(\mathbf{x}_Q) \end{bmatrix}$$

Matrix-free operator:

$$v_K = G^T D_K B u_K$$

Matrix-Free Algorithm for Advection

Face contribution

$$a_K^{\text{face}}(u, v) = \sum_{\hat{F} \in \partial \hat{K}} \sum_q \omega_q |J(\hat{\mathbf{x}}_q)| \; \textbf{upwind}\Big(\mathbf{a}(\mathbf{x}_q), J^{-T}(\hat{\mathbf{x}}_q)\hat{\mathbf{n}}, \hat{u}(\hat{\mathbf{x}}_q)\Big) \cdot [\![\hat{v}]\!](\hat{\mathbf{x}}_q)$$

Express the face operator as a sequence of operations:

$$v_K = \sum_{\hat{F} \in \partial K} B_{\hat{F}}^T D_{K,\hat{F}} \left(B_{\hat{F}} u_K, \; \tilde{B}_{\hat{F}} P_{K,\hat{F}} u_{\mathcal{N}_{\hat{F}}(K)} \right)$$

where

$$D_{K,\hat{F}}(u_K(\hat{\mathbf{x}}_q), u_{\mathcal{N}_{\hat{F}}(K)}(\hat{\mathbf{x}}_q)) = \omega_q |J(\hat{\mathbf{x}}_q)| \textbf{upwind}\Big(\mathbf{a}(\mathbf{x}_q), J^{-T}(\hat{\mathbf{x}}_q)\hat{\mathbf{n}}, u_K(\hat{\mathbf{x}}_q), u_{\mathcal{N}_{\hat{F}}(K)}(\hat{\mathbf{x}}_q)\Big)$$

Reducing the FLOPs: The sum factorization trick

On tensor product finite elements, the B operator can be computed as:

$$v_{k_1\dots k_6} = B_{IK}\,u_I = \underbrace{\sum_{i_1,\dots,i_6} u_{i_1\dots i_6}\,\varphi_{i_1}(x_{k_1})\dots\varphi_{i_6}(x_{k_6})}_{O(p^{12})=O(p^{2d})}$$

$$= \sum_{i_6}\varphi_{i_6}(x_{k_6})\Bigg(\dots\Big(\underbrace{\sum_{i_1}\varphi_{i_1}(x_{k_1})\,u_{i_1\dots i_6}}_{O(p^7)=O(p^{d+1})}\Big)\Bigg)$$

$$= \tilde{B}_{i_6 k_6}\otimes\dots\otimes\tilde{B}_{i_1 k_1}\,u_{i_1\dots i_6}$$

Roofline Model for Operator Evaluation

Benchmark Problem

$$v = Au$$

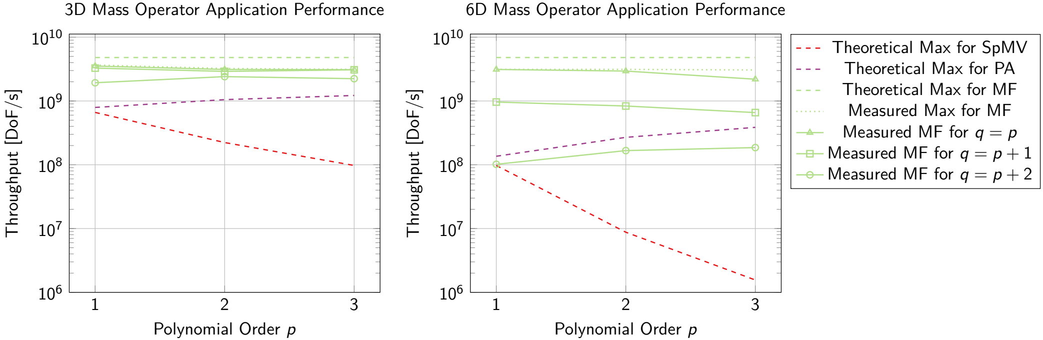
Theoretical data movements for the advection operator

Dimension	3D				6D			
Polynomial order	0	1	2	3	0	1	2	3
Number of dofs per element	1	8	27	64	1	64	729	4096
SpMV – Bytes per dof	104	260	560	1076	176	980	9740	52244
PA – Bytes per dof	784	259	158.2	119.1	21520	2749.7	1094.8	638.6
MF – Analytical mesh – Bytes per dof	16	16	16	16	16	16	16	16
MF – Linear space mesh – Bytes per dof	208	40	23	19	208	19	16.3	16.1
MF – Quadratic space mesh – Bytes per dof	664	97	40	26.1	664	26.1	16.9	16.2

SpMV: Sparse-Matrix vector product, PA: Partial Assembly operator application, MF: Fully Matrix-Free operator application

Performance Benchmark: Mass Operator

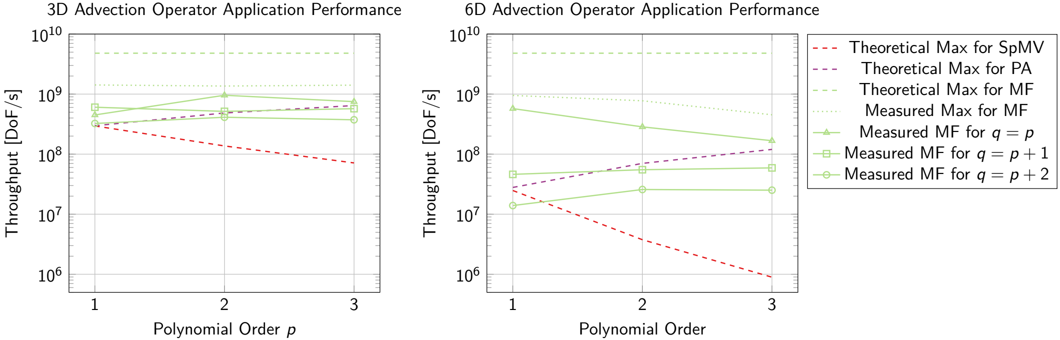
CPU Machine: Quartz (OpenMP - 76.8GB/s)



SpMV: Sparse-Matrix vector product, PA: Partial Assembly operator application, MF: Fully Matrix-Free operator application
p is the polynomial order, q is the number of quadrature points per dimension.

Performance Benchmark: Advection Operator

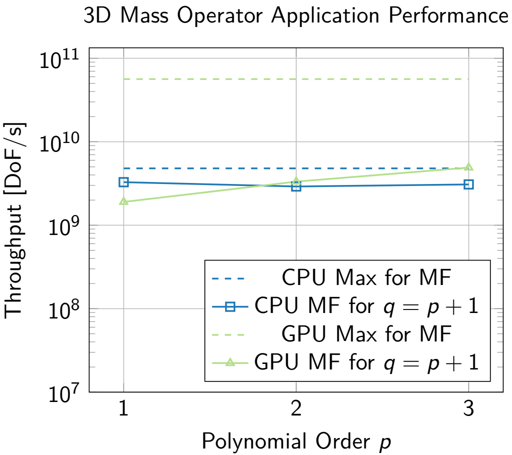
CPU Machine: Quartz (OpenMP - 76.8GB/s)

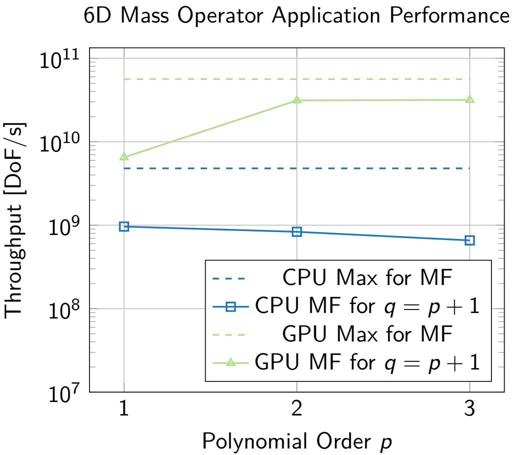


SpMV: Sparse-Matrix vector product, PA: Partial Assembly operator application, MF: Fully Matrix-Free operator application
p is the polynomial order, q is the number of quadrature points per dimension.

Early GPU performance results (Amit Rotem)

CPU Machine: Quartz (OpenMP - 76.8GB/s), GPU Machine: Lassen (V100 - 900GB/s)





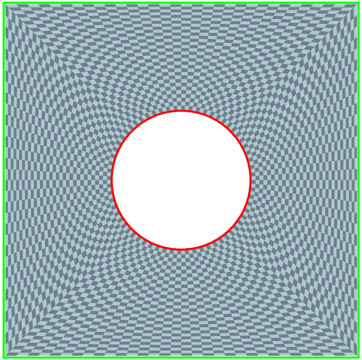
Main Takeaway:

Matrix-Free mass operator throughput is **higher** in 6D than 3D on GPU!

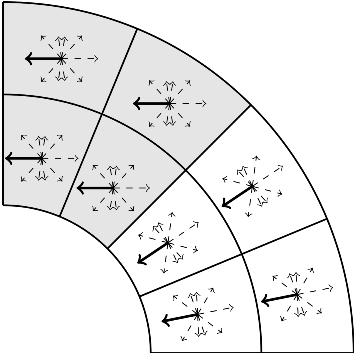
Rays-Effect Benchmark Problem I

Source Term:

- – Inflow function: $S_{in}(x, y, \omega) = 2$,
- – Outflow only.



(a) Spatial Mesh



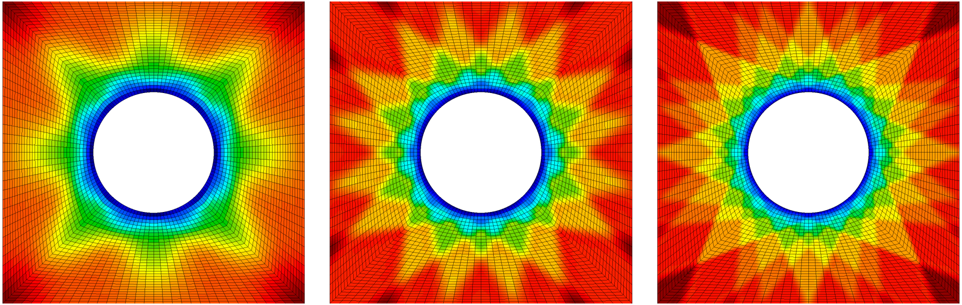
(b) Standard vs Cylindrical GSN

Figure: 2D Cartesian Space + 1D Polar Angle Problem.

Standard High-Order DG on Benchmark Problem I

Scalar flux:

$$F(X) = \int u(X,\omega)d\omega$$

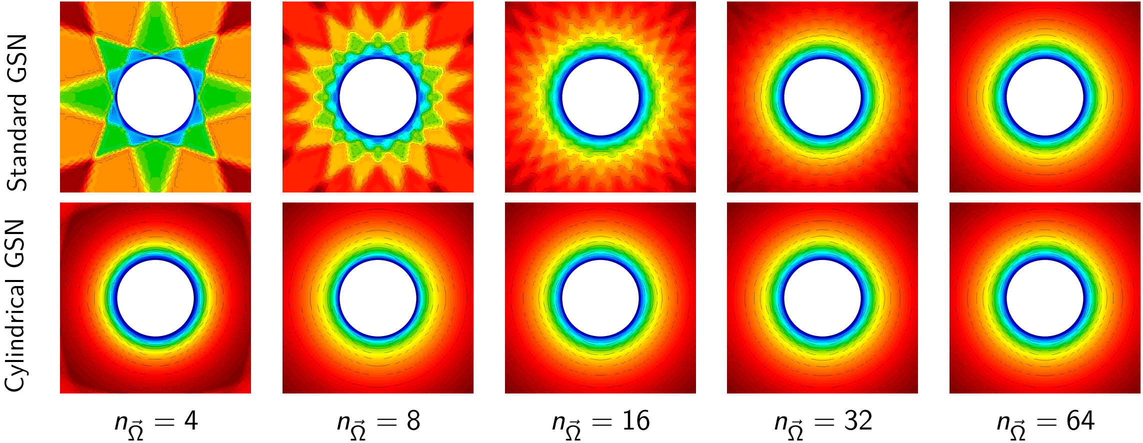


(a) Zeroth Order DG (b) First Order DG (c) Second Order DG

Figure: High order DG methods do not solve the rays-effect.

Comparing Standard with Cylindrical GSN

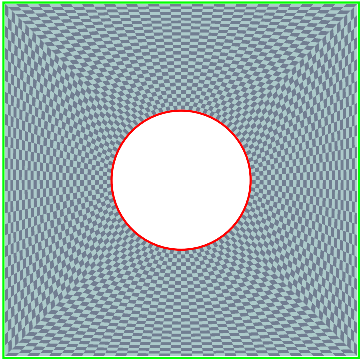
Benchmark Problem I: First Order



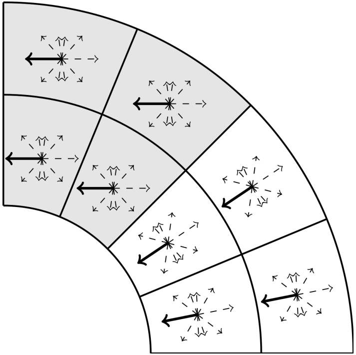
Rays-Effect Benchmark Problem II

Source Term:

- – Inflow function: $S_{in}(x, y, \omega) = (x^2 + y^2)^4$,
- – Outflow only.



(a) Spatial Mesh

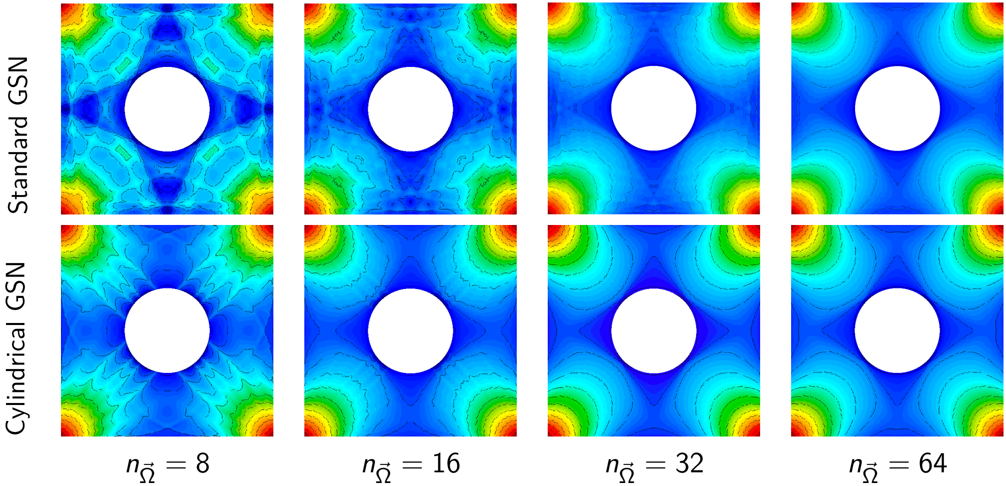


(b) Standard vs Cylindrical GSN

Figure: 2D Space + 1D Angle Problem.

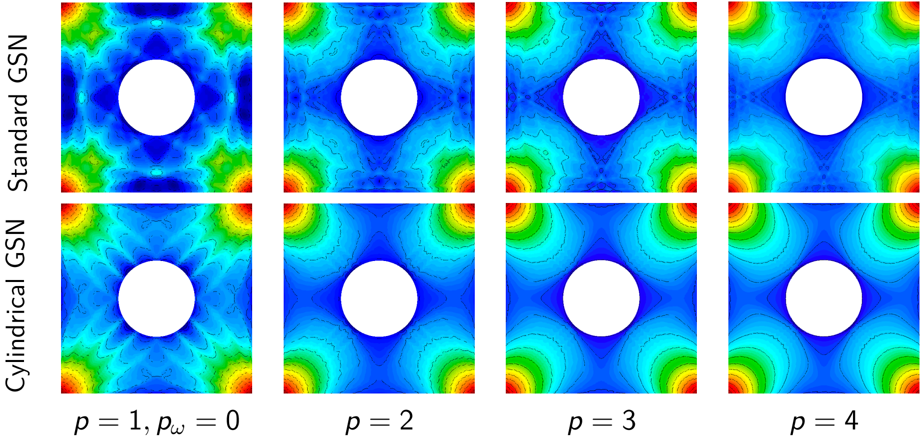
Comparing "Standard" with "Polar" Coordinates

Benchmark Problem II: Second Order



Comparing Standard with Cylindrical GSN

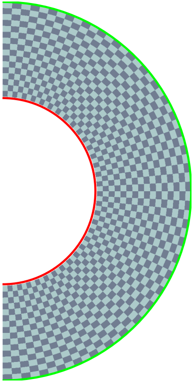
Benchmark Problem II: impact of the polynomial order p with $n_{\vec{\Omega}} = 16$



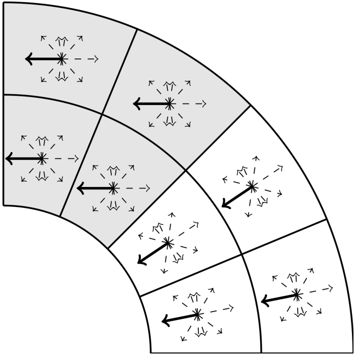
Rays-Effect Benchmark Problem III

Source Term:

- – Inflow function: $S_{in}(r, z, \omega, \phi) = 2$,
- – Outflow only.



(a) Spatial Mesh

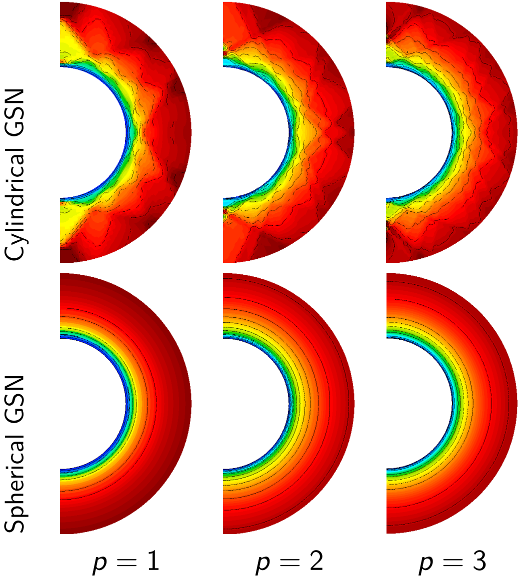


(b) Cylindrical vs Spherical GSN

Figure: 2D Space + 2D Angle Problem.

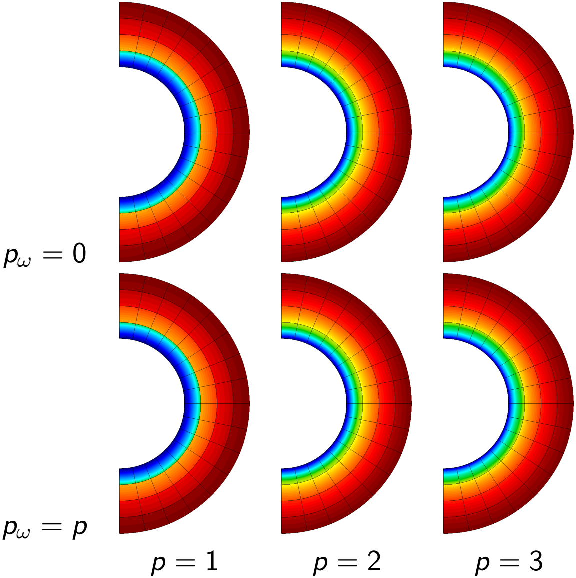
Benchmark III: Perfect Hohlraum in RZ spatial coordinates

Comparing "cylindrical" GSN with "spherical" GSN



Benchmark III: Perfect Hohlraum in RZ spatial coordinates

Comparing zeroth order in angle with high order in angle using spherical GSN



Beyond Analytical-GSN: Flux-GSN

Net flux (First Moment):

$$\vec{F}(X) = \int_{\omega} p(X,\omega)u(X,\omega)d\omega$$

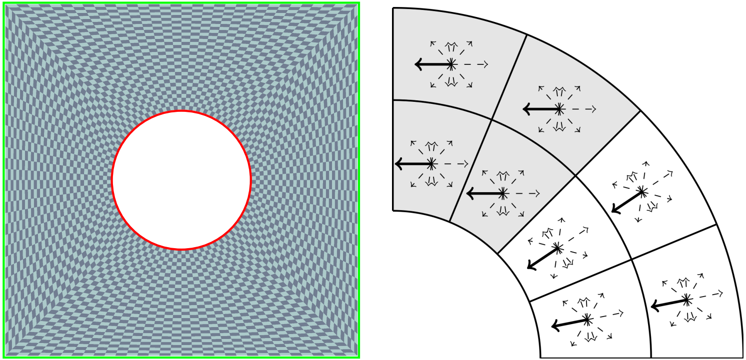
Flux-GSN algorithm:

1. Compute initial solution using analytical-GSN
2. Compute net flux
3. Fixed-point iteration until convergence
   1. Compute solution using flux-GSN with net flux: $\vec{\zeta}(X) = \vec{F}(X)$
   2. Compute net flux

Rays-Effect Benchmark Problem IV

Source Term:

- – Inflow function: $S_{in}(x, y, \omega) = 4 - x$,
- – Outflow only.

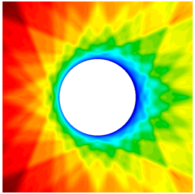


(a) Spatial Mesh (b) Polar Change of Coordinates

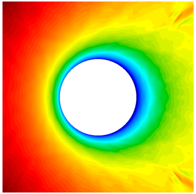
Figure: 2D Space + 1D Angle Problem.

Flux-GSN example

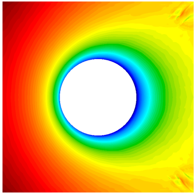
Benchmark Problem IV



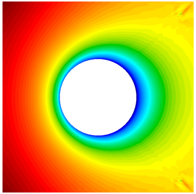
Initial solution



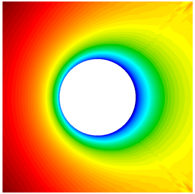
Iteration 0



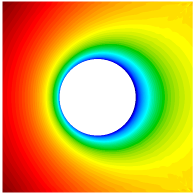
Iteration 1



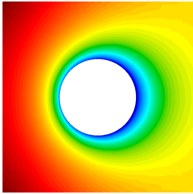
Iteration 2



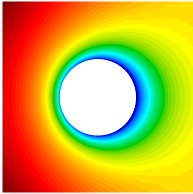
Iteration 3



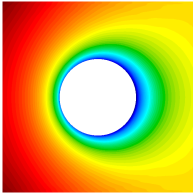
Iteration 4



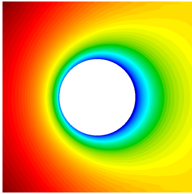
Iteration 5



Iteration 6



Iteration 7



Iteration 8

$\Rightarrow$ Converges to $1e^{-6}$ in 19 iterations.

Conclusion

Main Takeaways

- The Analytical-GSN and Flux-GSN methodologies can efficiently mitigate the rays-effect at low computational cost.
- Matrix-free algorithms mitigates efficiently the computational cost of high-dimensional simulation.
- The higher the dimension the higher the throughput on GPU architectures.
- Tensor product meshes enable easy construction of arbitrary dimension meshes.

Future Work

- Use other quantities than net flux to inform flux-GSN's coordinate system.
- Improvement of matrix-free solvers and preconditioners.
- Extension to non-conforming meshes (AMR).

Disclaimer

Lawrence Livermore
National Laboratory