Interpolation at Arbitrary Points in High-Order Meshes on GPUs

MFEM Community Workshop

22-24 October 2024

CASC
Center for Applied Scientific Computing

Ketan Mittal

Aditya Parik (USU), Tzanio Kolev (LLNL)

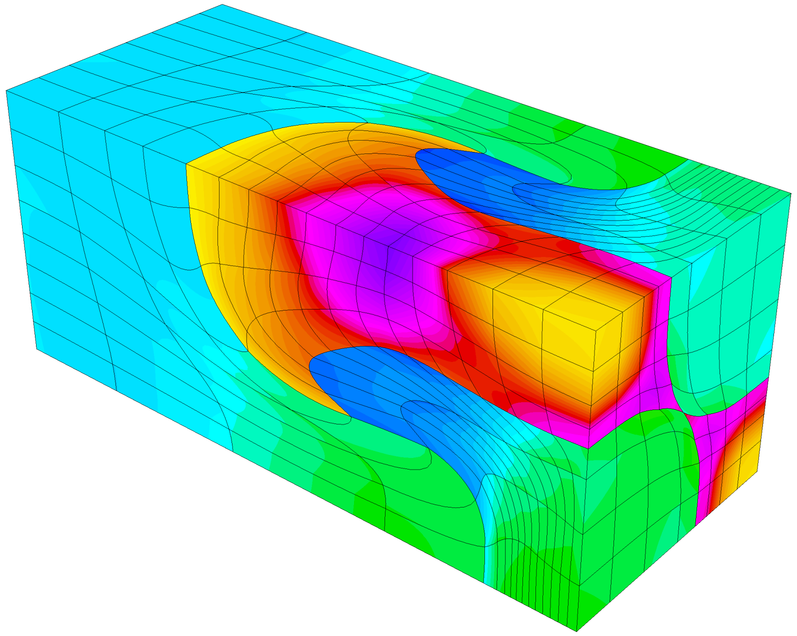
LLNL-PRES-870644

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

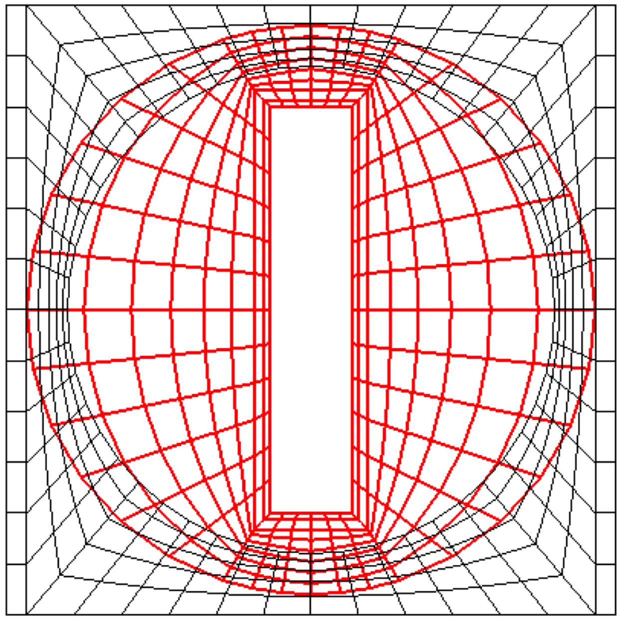
Lawrence Livermore National Laboratory

Motivation

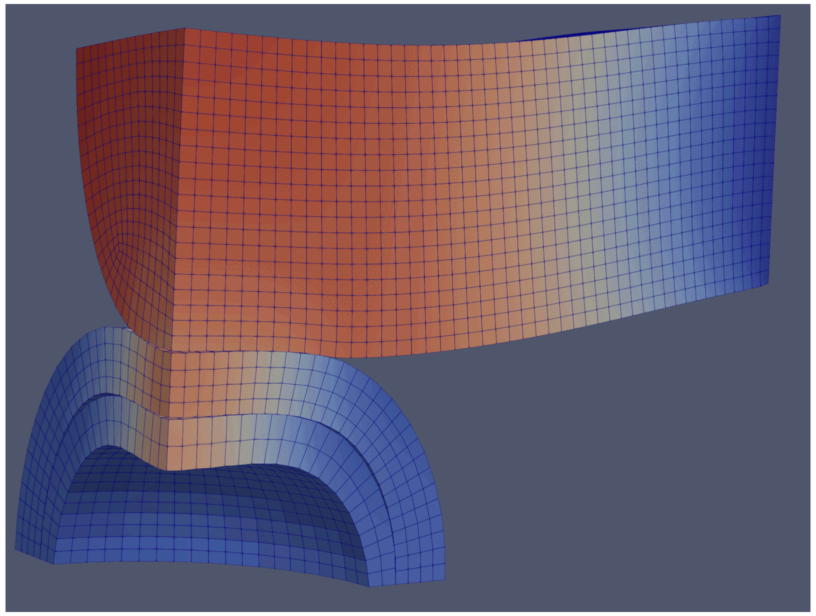
- Interpolation at arbitrary points is required in FEM for:
  - Querying the solution at desired locations.
  - Exchanging information between overlapping grids.
  - Detecting contact between meshes.



$p = 3$ mesh for triple-point problem



$p = 2$ meshes for overlapping grids

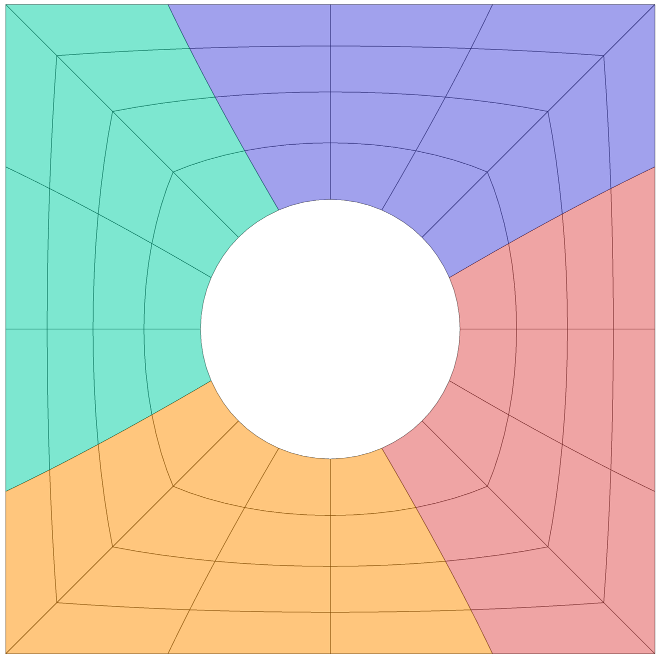


$p = 1$ meshes in contact

This is a challenging problem, especially for unstructured curvilinear meshes distributed on many MPI ranks in HPC.

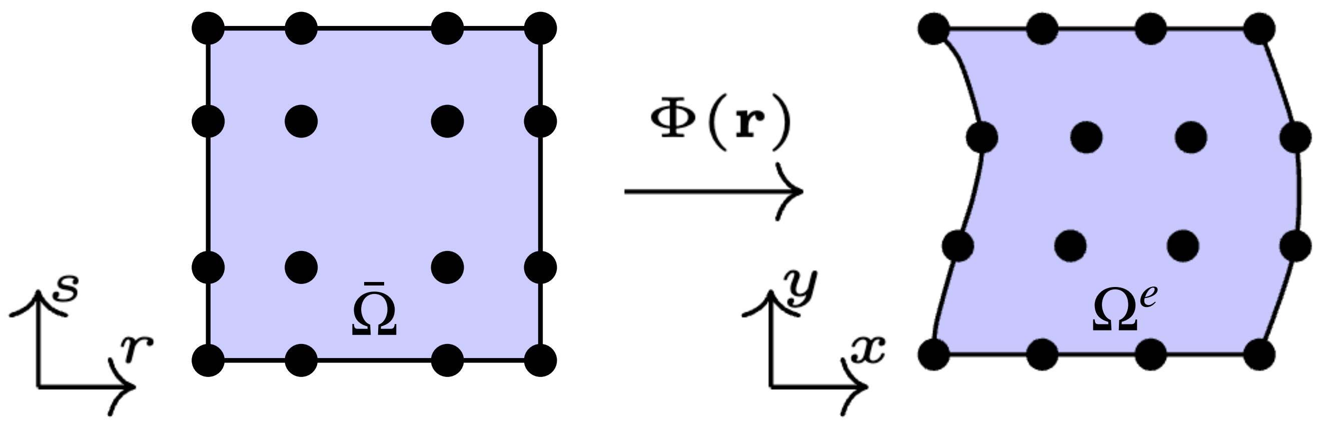
Background

- Each *tensor-product* element in the mesh is represented using Lagrange interpolants $\phi_i(\mathbf{r}), i = 1 \ldots N^D,\ D = [1,3],$ on Gauss-Lobatto-Legendre points in the reference element $\bar{\Omega} \in [0,1]^D$.



Quad mesh on 4 MPI ranks

- $$\mathbf{x}(\mathbf{r})\,|_{\Omega^e} = \sum_{l=1}^{N^D} \mathbf{x}_l^e \phi_l(\mathbf{r}) = \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N} \mathbf{x}_{ijk}^e \phi_i(r)\phi_j(s)\phi_k(t), \quad \mathbf{r} \in \bar{\Omega}$$



- Similar mapping is used for any function, e.g., Velocity $\mathbf{u}(\mathbf{r})$ and Temperature $T(\mathbf{r})$.

For a given point $\mathbf{x}^*$, we need to know the element e^* on MPI rank p^* that overlaps the point, and the corresponding reference-space coordinates $(\mathbf{r}^*)$ inside Ω^{e^*}.

Methodology

- **One time setup**
  - Compute data structures to quickly map a given point, $\mathbf{x}^* = \{x^*, y^*, z^*\}$, first to MPI ranks ($\underline{p}$) and then to candidate elements ($\underline{e}$) locally on each rank.
  - Compute element-wise bounding boxes to quickly test if a given point is inside an element.
- **Search a given set of points**
  - Newton search to compute reference-space coordinates in the candidate elements:

$$\underset{\mathbf{r}}{\mathtt{argmin}} \frac{||\mathbf{x}^* - \mathbf{x}(\mathbf{r})||_2^2}{2}$$

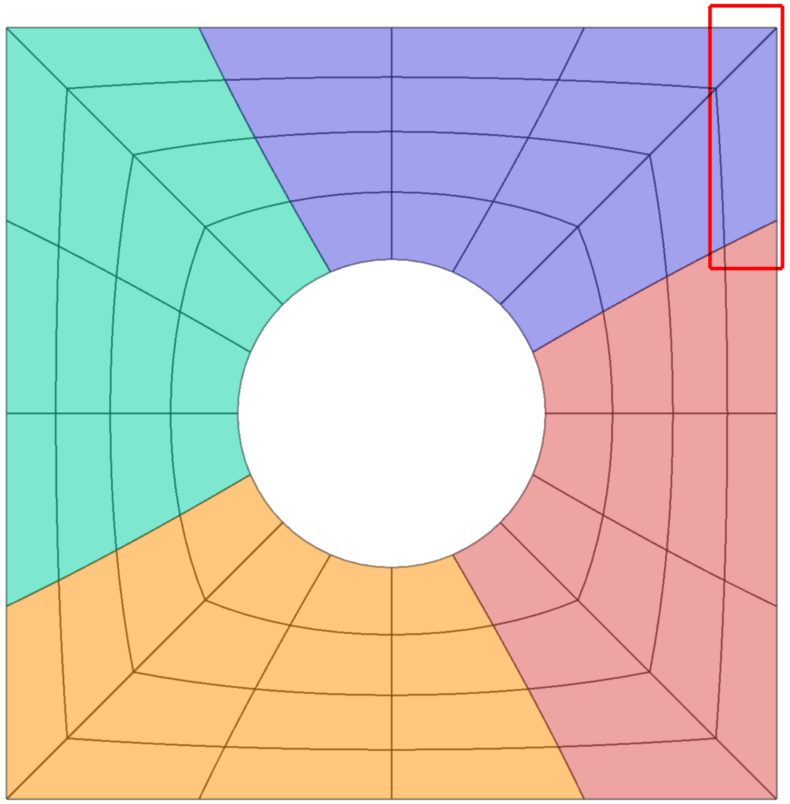
- **Interpolate the discrete solution**
  - $$u\left(\mathbf{x}^* = \Phi(\mathbf{r}^*)\right) = \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N} u_{ijk}^{e}\phi_i(r^*)\phi_j(s^*)\phi_k(t^*)$$

*Implementation based on **gslib** developed by James Lottes in the context of SEM for Nek5000*

Setup

Axis-Aligned Bounding Box

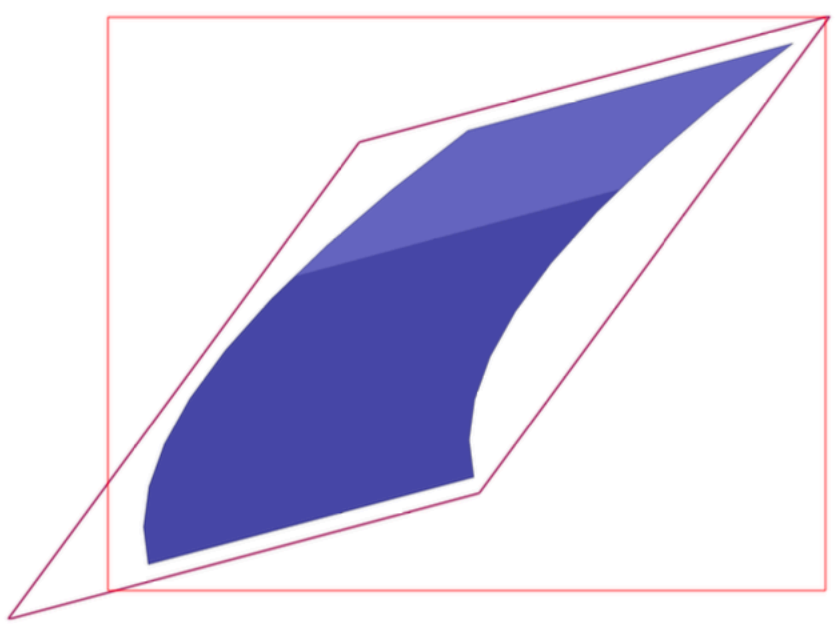
- Compute bounds on $\mathbf{x}(\mathbf{r})\,|_{\Omega^e} = \sum_{i=1}^{N}\sum_{j=1}^{N} \mathbf{x}^e_{ij}\phi_i(r)\phi_j(s)$ to determine $\{x_{\mathtt{min}}, x_{\mathtt{max}}, y_{\mathtt{min}}, y_{\mathtt{max}}\}$.



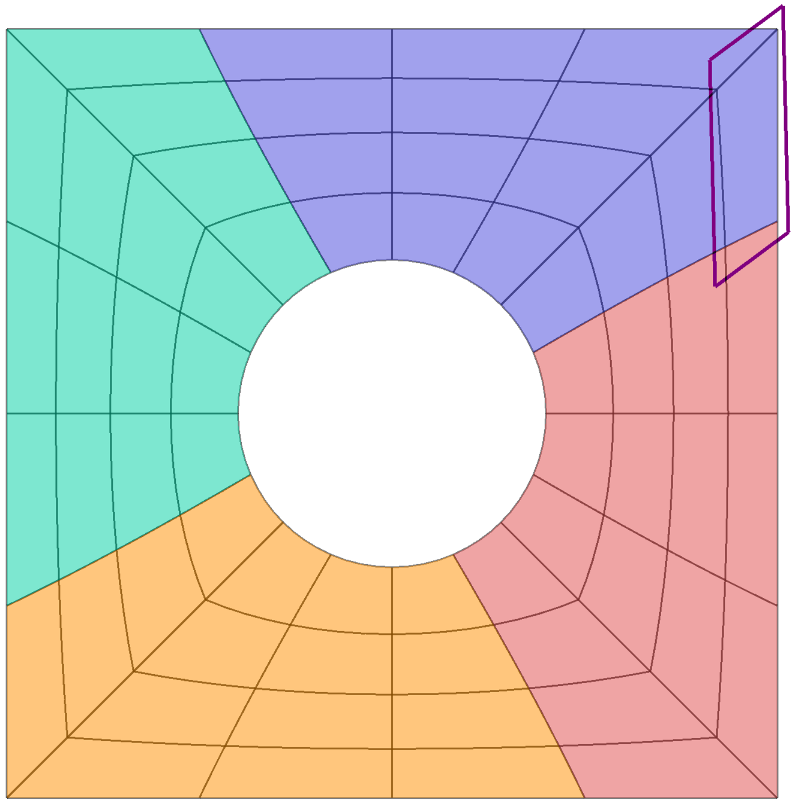
Axis-Aligned Bounding Boxes

Oriented Bounding Box

- AABB are suboptimal for curvilinear elements.
- OBB provide tighter bounds around each element.
  - Represented by OBB center and a transformation matrix ($A_{D \times D}$) with respect to the reference element.



Bounding boxes around a curvilinear element



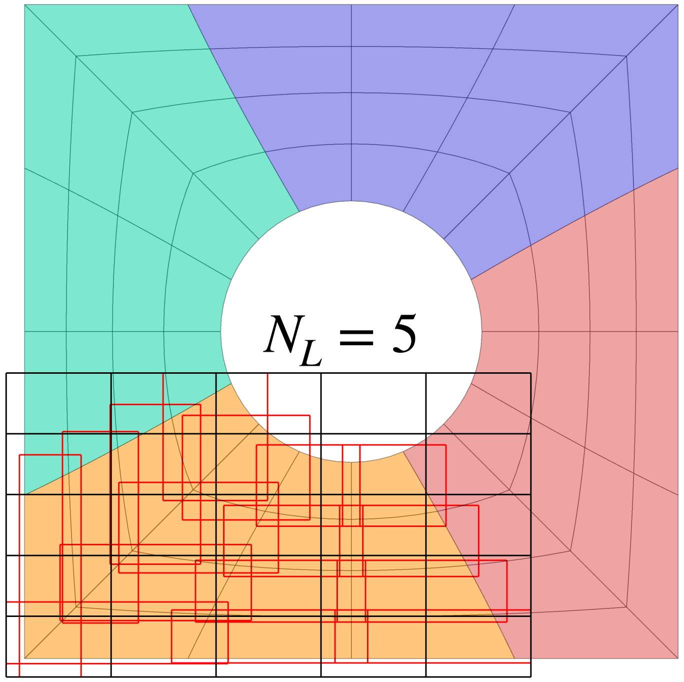
Oriented Bounding Boxes

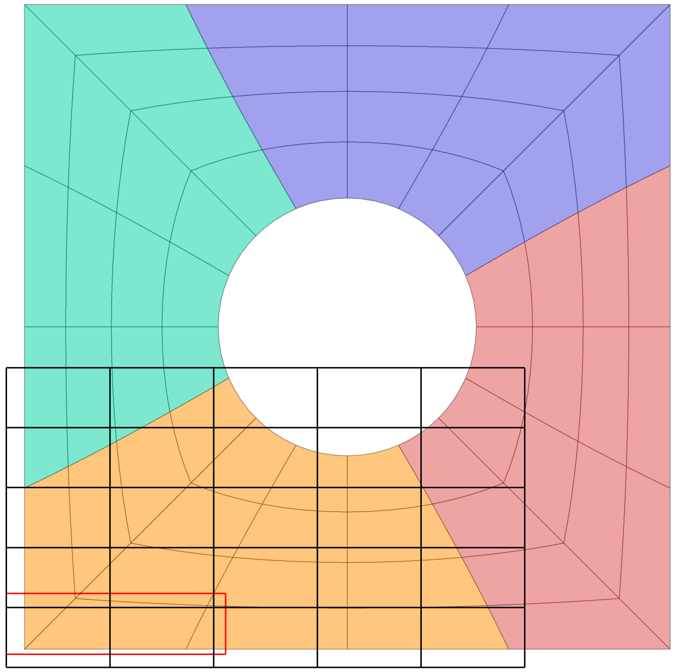
Local Map for $\mathbf{x}^* \to \{\underline{e}\}$

- Generate a uniform $N_L \times N_L$ Cartesian mesh ($\mathscr{M}_L$) over the domain of the union of processor-local AABB $\{x_{L,\min}, x_{L,\max}, y_{L,\min}, y_{L,\max}\}$.
  - $\mathscr{M}_L$ is never explicitly constructed.
- Compute intersection between **elements of $\mathscr{M}$ and $\mathscr{M}_L$**:

  $\mathtt{std::map<int\ e_{\mathscr{M}_L}, Array<int>\ e_{\mathscr{M}}>}$

  - Intersection of AABB and cells of $\mathscr{M}_L$ is trivial.
- Given $\mathbf{x}^*$, determine which cell of $\mathscr{M}_L$ it is located in, and look-up candidate element using the map.



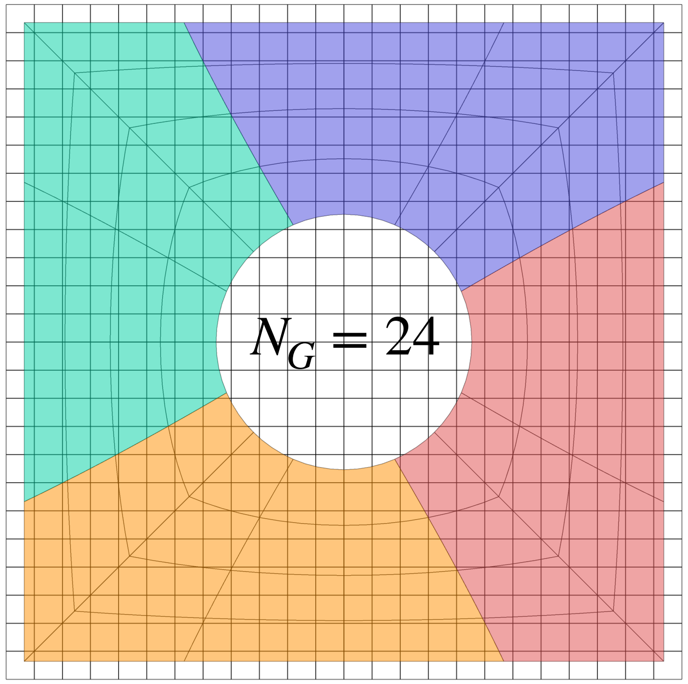


Global Map for $\mathbf{x}^* \rightarrow \{\underline{p}\}$

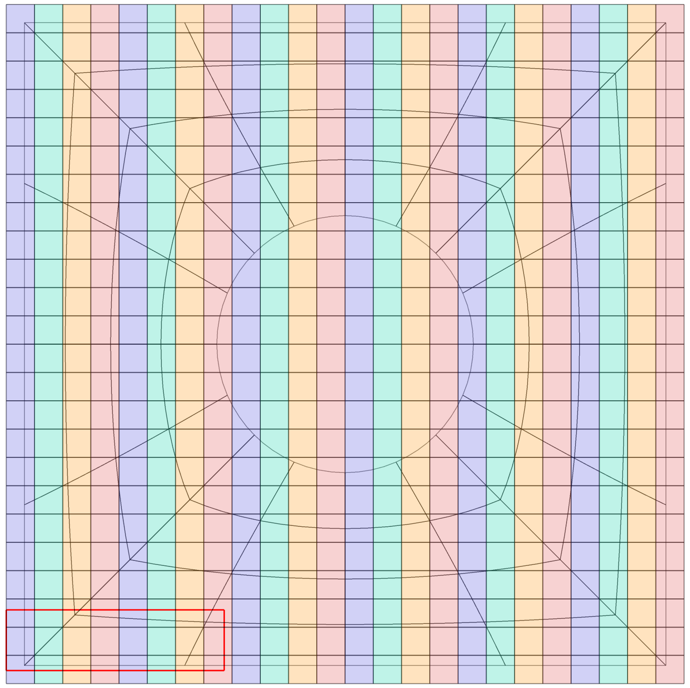
- Generate a uniform $N_G \times N_G$ Cartesian mesh ($\mathscr{M}_G$) over the entire mesh $\{x_{G,\min}, x_{G,\max}, y_{G,\min}, y_{G,\max}\}$.
- $\mathscr{M}_G$ is globally partitioned. Each rank checks intersection of its elements with elements of $\mathscr{M}_G$, and communicates to corresponding MPI ranks:

  $\mathtt{std::map<int\ e_{\mathscr{M}_G}, Array<int>\ p_{\mathscr{M}}>}$

- Given $\mathbf{x}^*$, determine which cell of $\mathscr{M}_G$ it is located in, and send to corresponding MPI rank. Then locally look-up the list of ranks that can contain the point and forward the point to those ranks.



Cartesian mesh overlapping the entire domain.



AABB for one of the elements of $\mathscr{M}$.

FindPoints

CPU↔GPU Data Movement

- Data Movement:

[input]

- Mesh nodes - $D \cdot N_E \cdot N^D$
  - *Should already be on GPU.*
- Bounding boxes - $(3D + D^2)N_E$
- Local hash mesh - $O(N_E^D)$
- Coordinates of points to be found - $D \cdot N_{pt}$

[output]

- Element index - N_{pt}
- Reference-space coordinates - $D \cdot N_{pt}$
- Distance between actual and found point - N_{pt}
- code (inside element/border/not found) - N_{pt}

- `Coalesced memory` access.
- `Shared memory` for fast data access of work arrays used with SIMD instructions.

Initial Element Look-up and Bounding Box Test

- For each point, look up candidate elements based on the hash table. Then check bounding boxes for each candidate elements:
  - AABB test
    - $(x_i^* - x_{i,\mathtt{min}}^e)(x_{i,\mathtt{max}}^e - x_i^*) > 0 \ \forall i \in [1,D], i \in \mathbb{Z}$
  - OBB test
    - $-1 \leq A_{D\times D}^{-1}(\mathbf{x}^* - \mathbf{x}_c) \leq 1$, where $A_{D\times D}$ captures the OBB size and orientation, and $\mathbf{x}_c$ is the OBB center.

FindPoints - Newton's Method

- Minimize $f(\mathbf{r}) = \frac{1}{2}||\mathbf{x}^* - \mathbf{x}(\mathbf{r})||_2^2 = \frac{1}{2}||\Delta x_i||_2^2$ using Newton's method with trust-region.
  - $\mathbf{r}_0$ based on closest mesh node.
  - $\mathbf{r}_{l+1} = \mathbf{r}_l - \underbrace{\mathscr{H}_{ji}^{-1}\mathscr{J}_j}_{\Delta\mathbf{r}_l}, \quad \mathbf{r}_{l+1} \in [-1,1]^D, \ \ \Delta\mathbf{r}_l \in \alpha_l[-1,1]^D.$

$$\mathscr{J}_j = \frac{\partial f}{\partial r_j} = \sum_i \Delta x_i \left(-\frac{\partial x_i}{\partial r_j}\right) = -G_{ij}^T \Delta x_i, \qquad \mathscr{H}_{jk} = \frac{\partial^2 f}{\partial r_j \partial r_k} = G_{ji}G_{ik} - \beta \Delta x_i \frac{\partial^2 x_i}{\partial r_j \partial r_k}, \qquad G_{ij} = \frac{\partial x_i}{\partial r_j}$$

  - $\beta = 0$ when searching inside the element, 1 if searching on element edge/face.
  - α_l is a trust-region factor that depends on the quality of most recent Newton update.

Newton's Method - Searching interior to an element

- Ignoring the second derivative term in the Hessian simplifies the Newton update to

  $\mathbf{r}_{l+1} = \mathbf{r}_l + G_{ij}^{-1} \Delta x_i.$

  - Requires evaluation of $\Delta x_i(\mathbf{r}_l) = x_i^* - x_i(\mathbf{r}_l)$ and $G_{ij}(\mathbf{r}_l) = \dfrac{\partial x_i(\mathbf{r}_l)}{\partial r_j}.$

- We use N_{pt} thread blocks (1 for each point to be found), and each block has $N \cdot D$ threads.

  - Tensor-product structure is leveraged to use 1D operator, which maps well to the N threads.

SIMD Instructions for Parallelizing Work

- First evaluate basis functions and their derivatives:
  - N threads - $\phi_i(r)$ and $\dfrac{\partial \phi_i(r)}{\partial r}$.
  - N threads - $\phi_i(s)$ and $\dfrac{\partial \phi_j(s)}{\partial s}$.
- Next, evaluate the inner summation:
  - N threads - $x_i = \sum_{j=1}^{N} x_{ij}^e \phi_i(r)\phi_j(s)$. Same for $\dfrac{\partial x_i}{\partial r}, \dfrac{\partial x_i}{\partial s}$.
  - N threads - $y_i = \sum_{j=1}^{N} y_{ij}^e \phi_i(r)\phi_j(s)$. Same for $\dfrac{\partial y_i}{\partial r}, \dfrac{\partial y_i}{\partial s}$.
- Finally, thread 0 accumulates the outer summation.

$$x^e(\mathbf{r}) = \sum_{i=1}^{N}\sum_{j=1}^{N} x_{ij}^e \phi_i(r)\phi_j(s), \quad y^e(\mathbf{r}) = \sum_{i=1}^{N}\sum_{j=1}^{N} y_{ij}^e \phi_i(r)\phi_j(s)$$

$$\frac{\partial x}{\partial r} = \sum_{i=1}^{N}\sum_{j=1}^{N} x_{ij}^e \frac{\partial \phi_i(r)}{\partial r}\phi_j(s), \quad \frac{\partial y}{\partial r} = \sum_{i=1}^{N}\sum_{j=1}^{N} y_{ij}^e \frac{\partial \phi_i(r)}{\partial r}\phi_j(s)$$

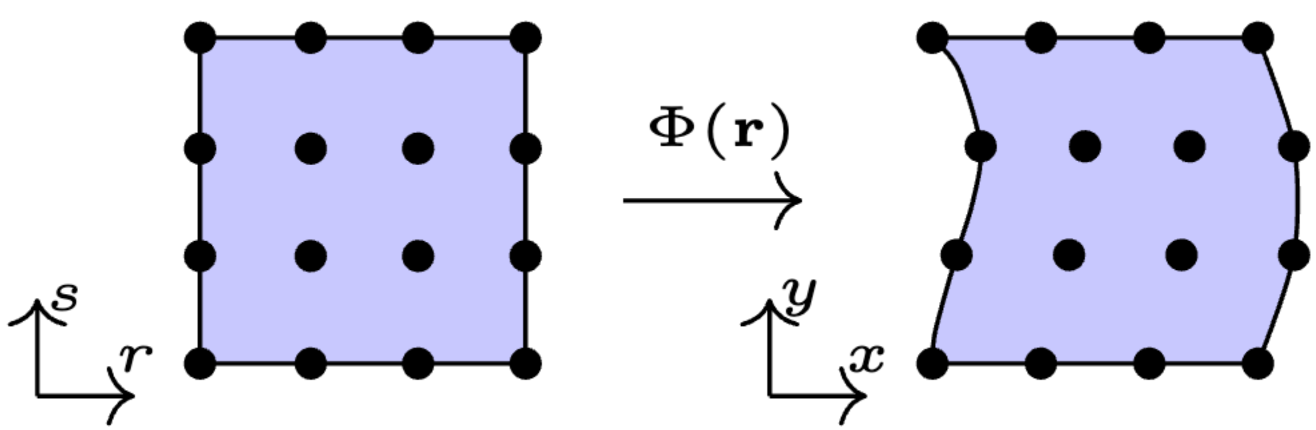
$$\frac{\partial x}{\partial s} = \sum_{i=1}^{N}\sum_{j=1}^{N} x_{ij}^e \phi_i(r)\frac{\partial \phi_j(s)}{\partial s}, \quad \frac{\partial y}{\partial s} = \sum_{i=1}^{N}\sum_{j=1}^{N} y_{ij}^e \phi_i(r)\frac{\partial \phi_j(s)}{\partial s}$$

Searching on Element Face/Edge

- After the first Newton iteration, we check $\mathbf{r}_l$ to search either inside an element, on the face (in 3D), or on the edge of the element.

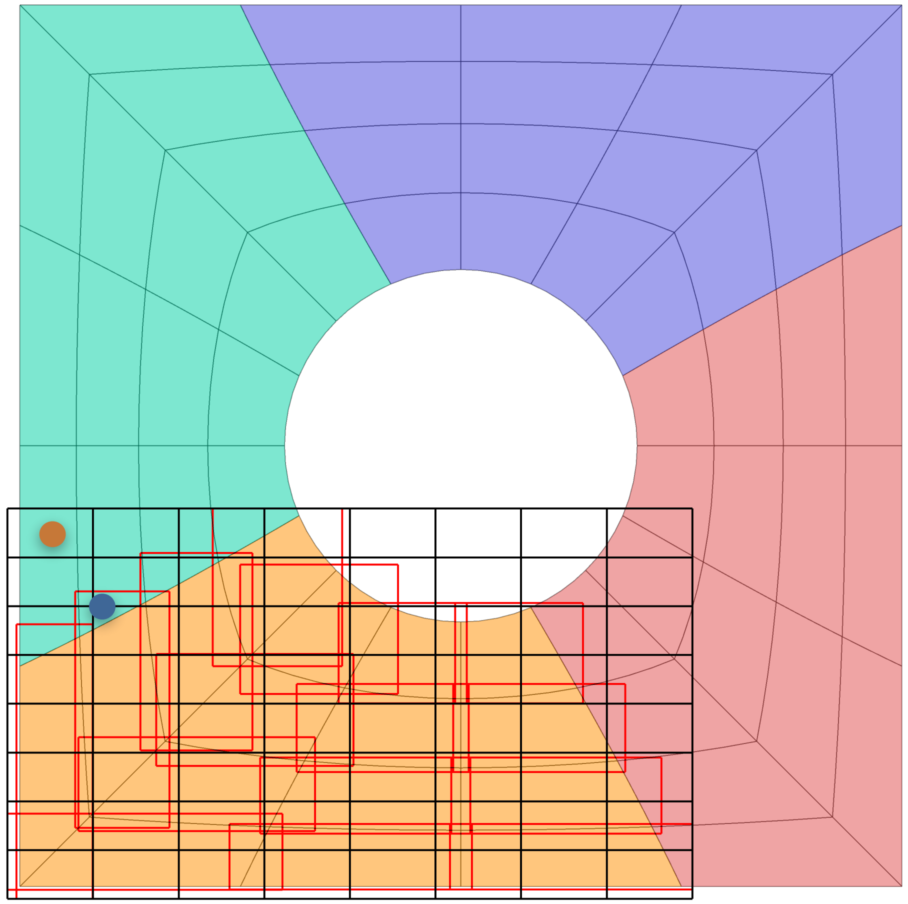
- Fewer unconstrained variables when searching on face (2 in 3D) or edge (1).

  - For example, $\mathscr{J} = \frac{\partial f}{\partial r}$ and $\mathscr{H} = \frac{\partial^2 f}{\partial^2 r}$ for edge corresponding to $s \pm 1$.



Logic for Points Not Found Locally

- Points that are not found in the local mesh are routed to other MPI ranks using the global map.
- For points found outside the element, we use element that returns minimum $||\Delta\mathbf{x}||$.



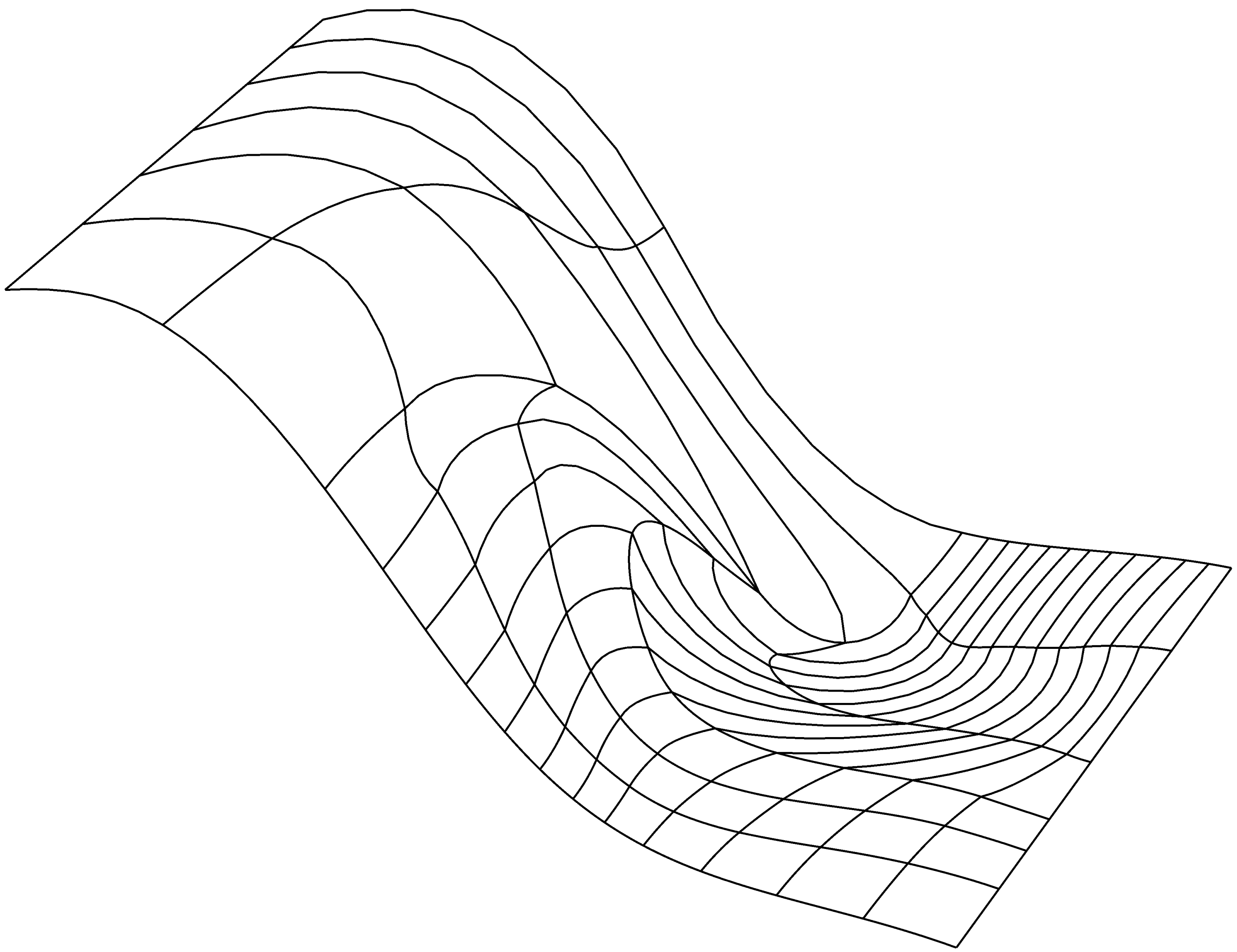
- Point inside the hash mesh but not in any element's bounding box.
- Point inside the bounding box but not the element

Interpolation

$$u(\mathbf{x}(\mathbf{r})) = u(\mathbf{r})\,|_{\Omega^e} = \sum_{i=1}^{\tilde{N}} \sum_{j=1}^{\tilde{N}} \sum_{k=1}^{\tilde{N}} u_{ijk}^e \phi_i(r)\phi_j(s)\phi_k(t),$$

- N_{pt} blocks, 1 for each point, with $\tilde{N}^d$ threads in each block.
- Interpolation done first for processor local points, and then for points originating on other ranks but owned by elements on current rank.

Extension to Surface Meshes

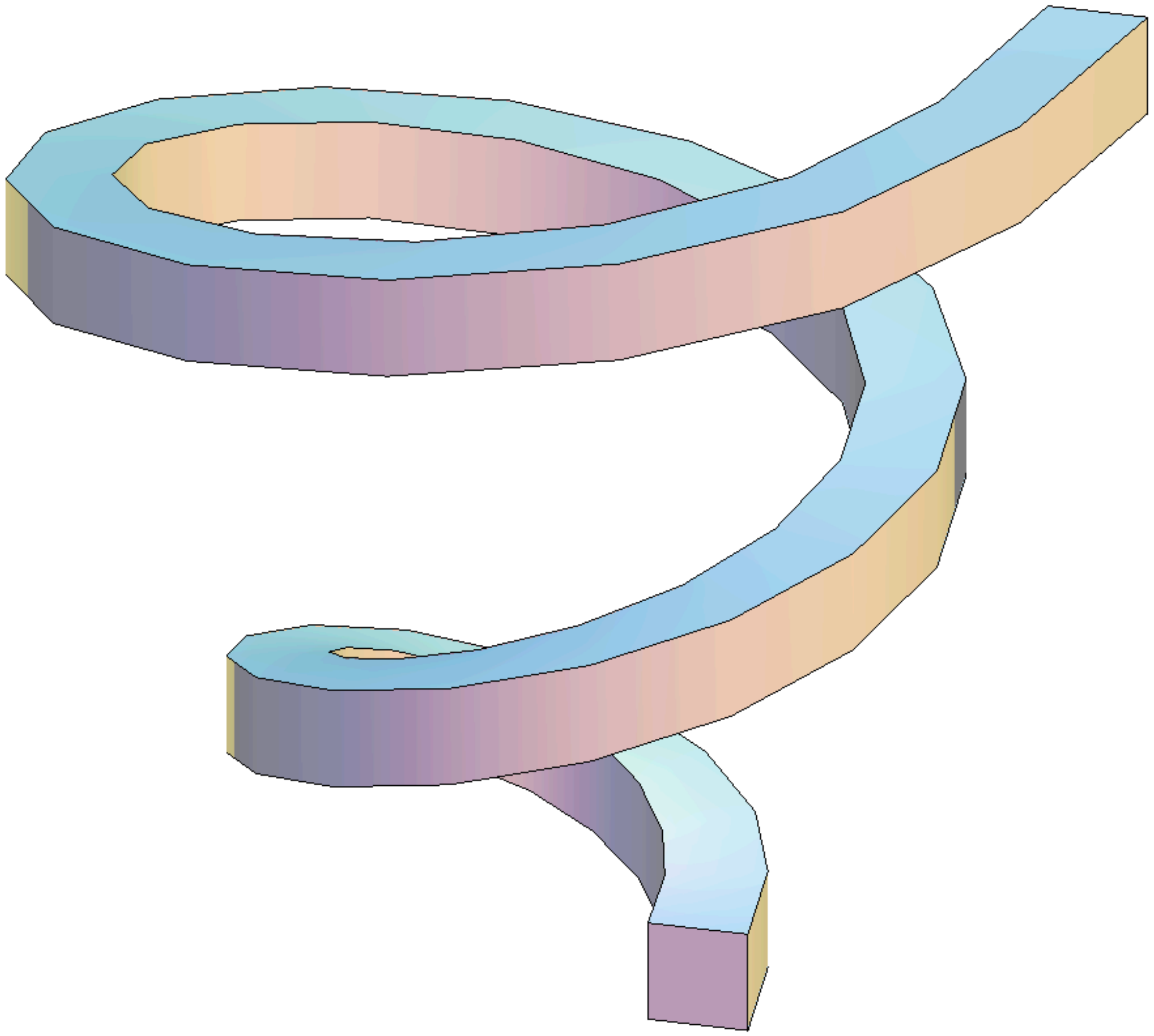


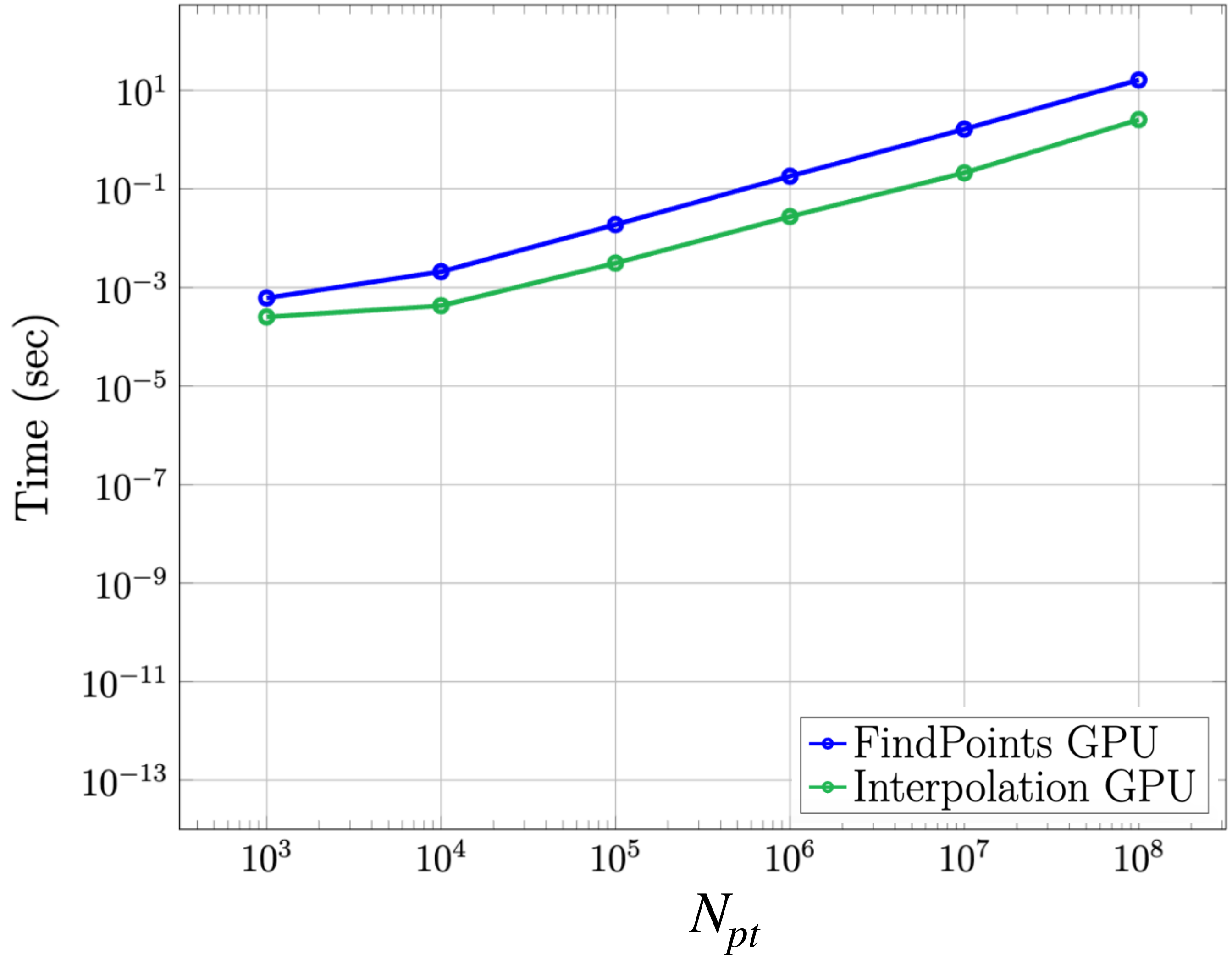
Aditya Parik's talk at 1:20 PM

Results

Spiral ($p = 9$)

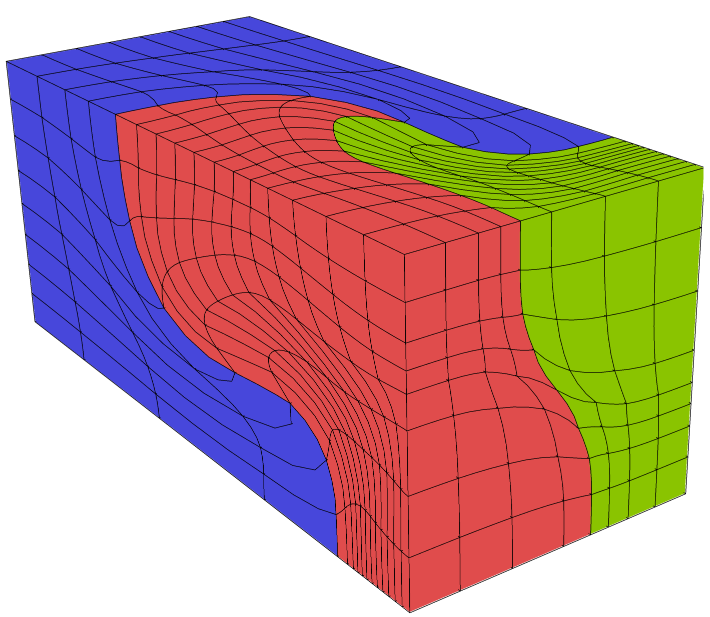
- Time to find up-to 100 million points in a 9th order element on 1 GPU (`miniapps/meshing/pfindpts`).
- 5 Newton iterations per point on average.

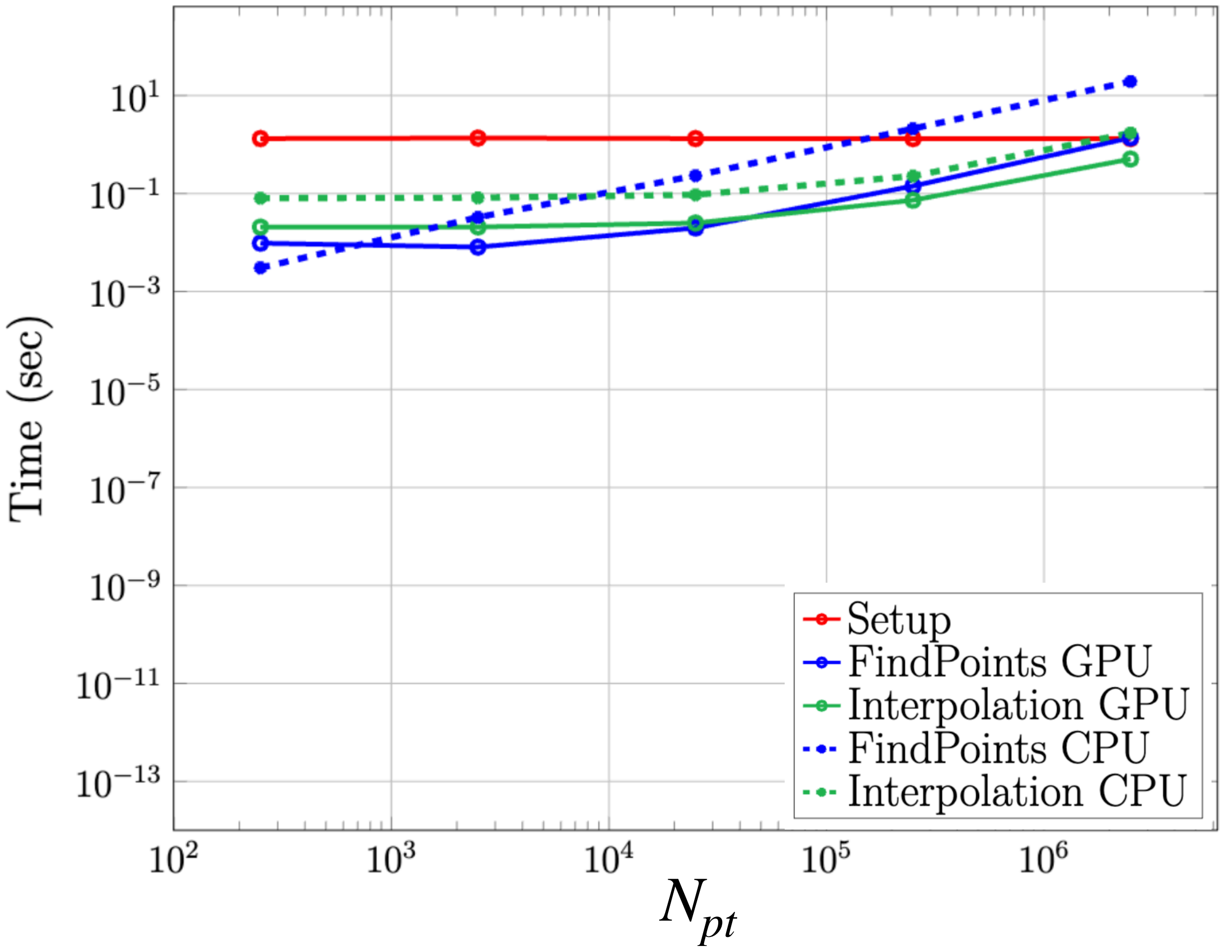




Triple Point Problem ($p = 3$)

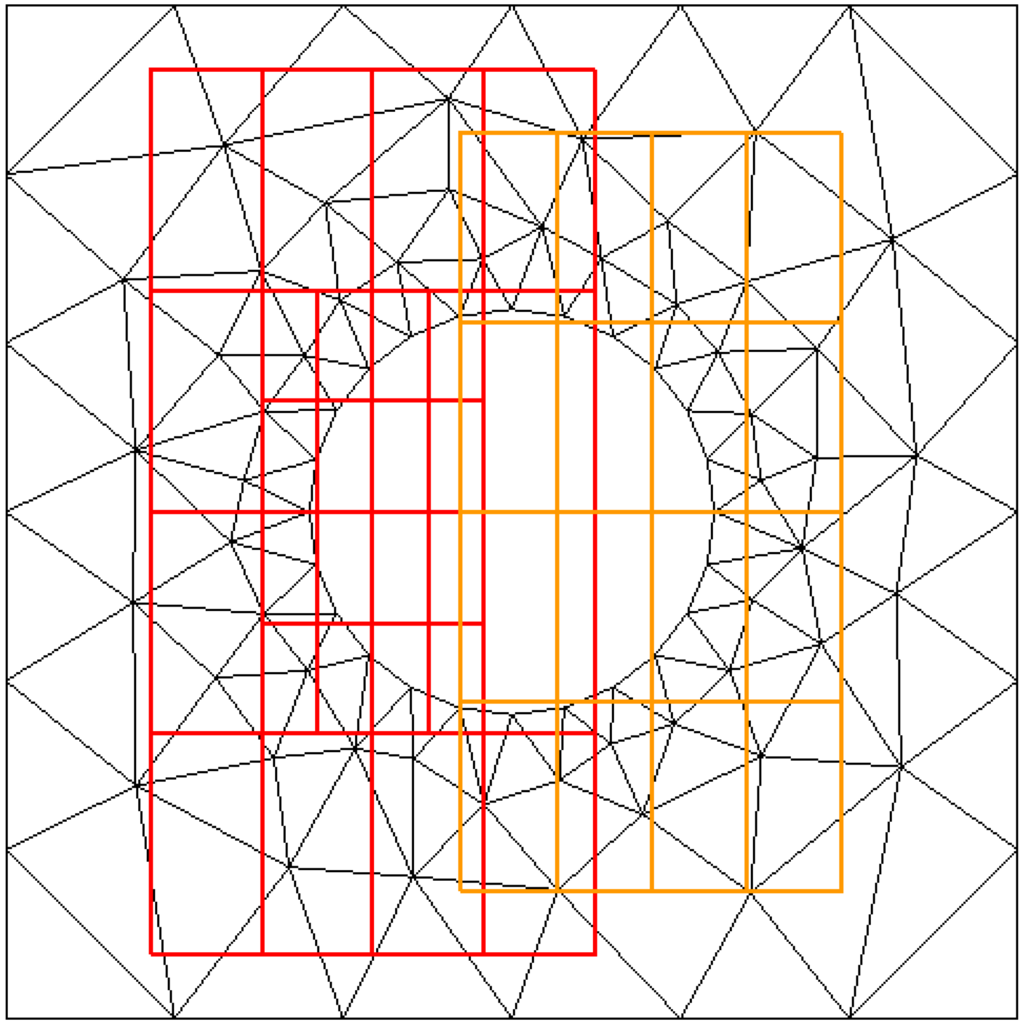
- $N_E = 65{,}536$, $p = 3$ for the triple point problem on 4 GPUs.
- Lassen supercomputer @ LLNL
  - 756 Nodes: 40 IBM Power9 CPU Cores + 4 Nvidia V100 GPUs per Node.
  - In GPU mode, we typically run on 4 GPUs + 4 CPU cores with all computation on GPUs.



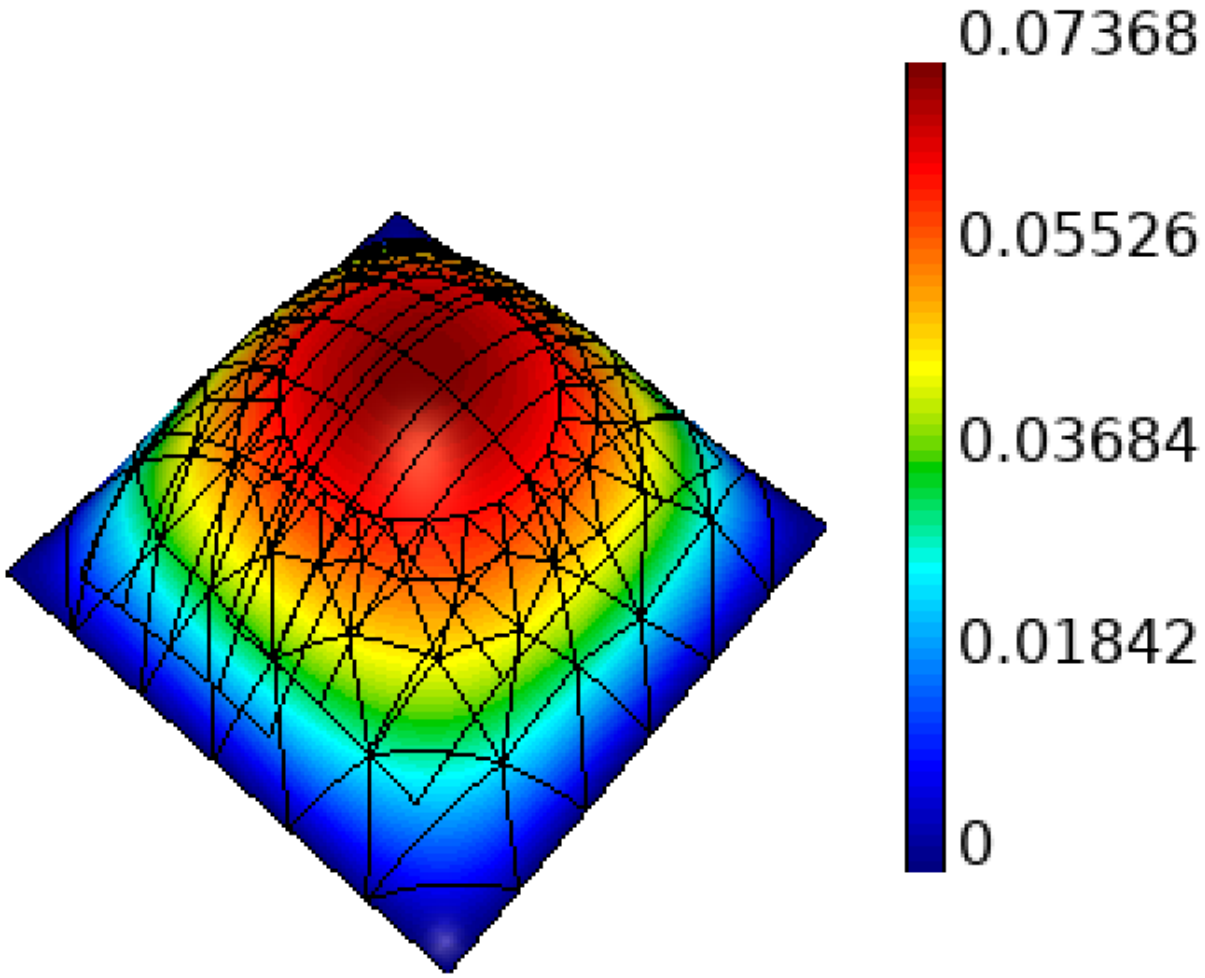


Applications - Solving PDEs on Overlapping Grids

- `miniapps/gslib/schwarz_ex1p`



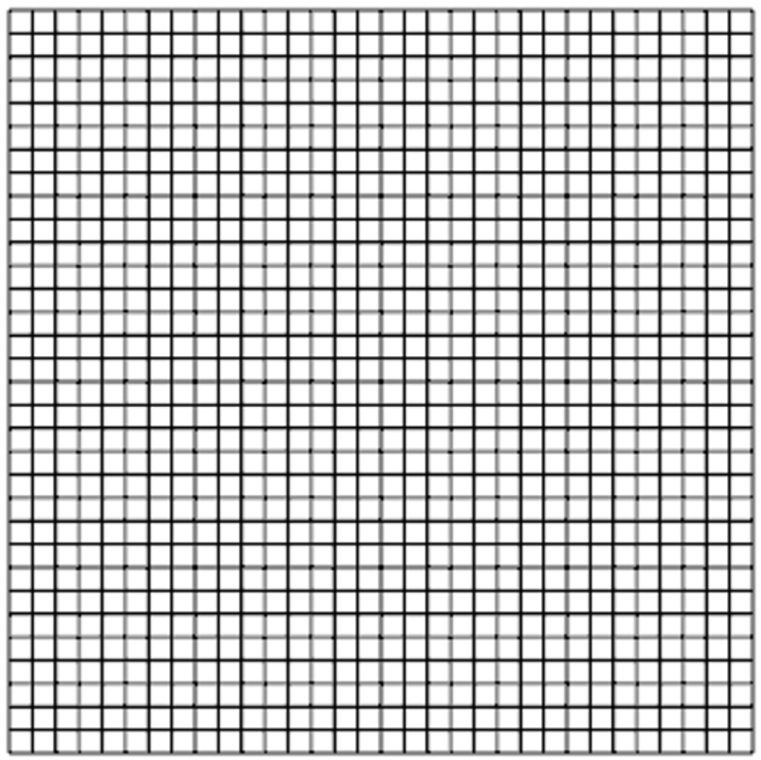
Overlapping grids



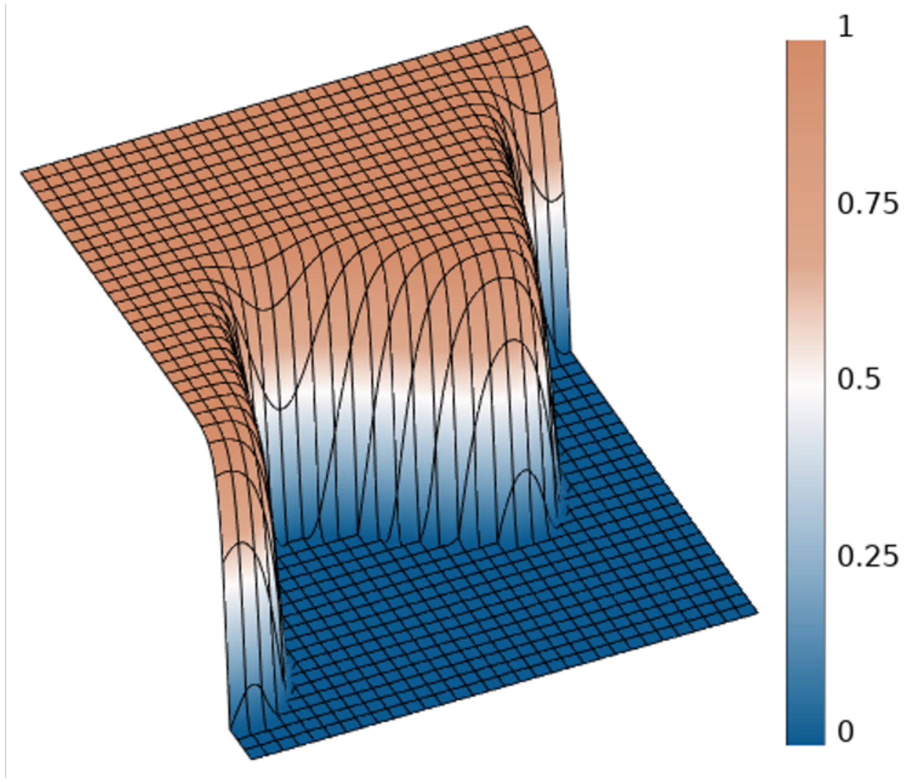
$-\nabla^2 u = f$ *on overlapping grids*

Applications - Mesh to Mesh Remap during r-adaptivity

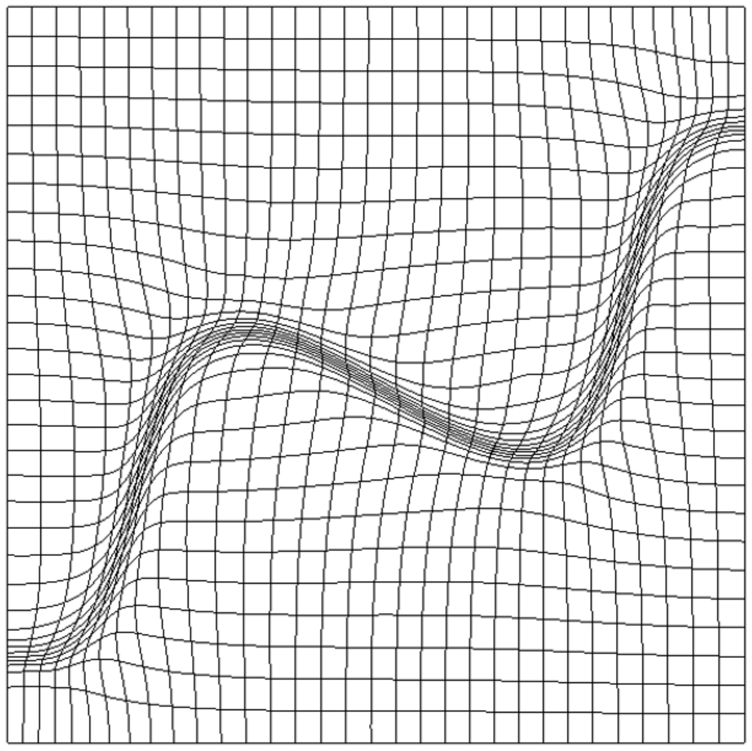
- `miniapps/meshing/pmesh – optimizer`



Initial mesh



Discrete function on initial mesh



Mesh adapted to discrete function with GSLIB-based remap during mesh optimization.

Some Remarks on Usage!

- All MPI ranks must call the methods simultaneously.
- Do not duplicate list of points on all the ranks.
- New methods enable custom interpolation where we do not directly use a GridFunction.

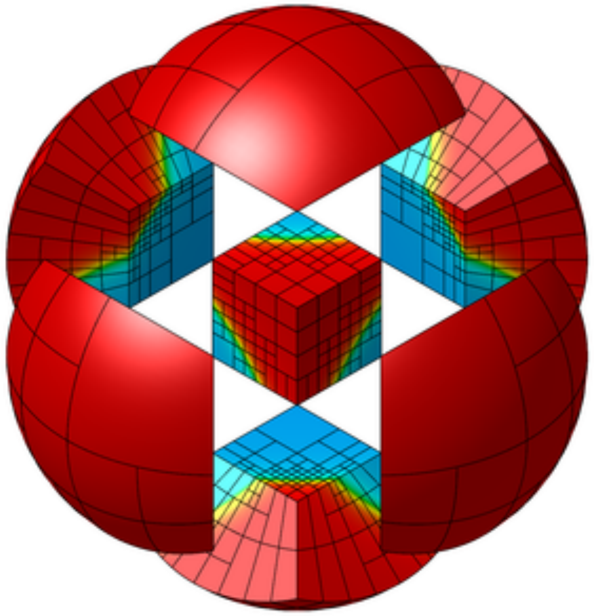
```
FindPointsGSLIB finder(MPI_COMM_WORLD);
finder.Setup(pmesh);
if ((myid != 0)) { xyz.Destroy(); }
finder.FindPoints(xyz, point_ordering);
finder.Interpolate(gf_in, interp_xyz);
```

```
FindPointsGSLIB finder;
finder.Setup(pmesh);
finder.FindPoints(xyz, Ordering::byVDIM);

/** Interpolate gradient using custom interpolation procedure. */
// We first send information to MPI ranks that own the element corresponding
// to each point.
Array<unsigned int> recv_elem, recv_code;
Vector recv_rst;
finder.DistributePointInfoToOwningMPIRanks(recv_elem, recv_rst, recv_code);
int npt_recv = recv_elem.Size();
// Compute gradient locally
Vector grad(npt_recv*dim);
.
.
.
// Send the computed gradient back to the ranks that requested it.
Vector recv_grad;
finder.DistributeInterpolatedValues(grad, dim, Ordering::byVDIM, recv_grad);
```

Summary & Future Work

- Thanks to Yohann Dudouit for the discussions.
- Robust arbitrary point search in high-order meshes on GPUs.
- Future work will extend implementation to simplices on GPUs.
- Paper under preparation with all the technical details!



mfem.org

CASC

Center for Applied Scientific Computing

Lawrence Livermore National Laboratory