Liszt is a domain specific language that exposes a high-level interface for building mesh-based solvers of PDEs. This frees scientists from architecture-specific implementations and increases programmer productivity tenfold. Current PSAAP solvers are tied to a specific platform, while Liszt solvers are portable across architectures. Our compiler achieves this by using domain knowledge in its program analysis stage to produce high performance code for a variety of platforms.

Liszt has a stable implementation for finite difference methods with a fully functional MPI-based backend. Liszt now supports implicit methods by providing native sparse matrix operations, as used by our implementation of the Joe RANS solver. Program transformations for our GPU runtime are in development, and our preliminary GPU runtime provides explicit finite difference support. A full stack of debugging, visualization, and compiler tools is now available.

### Implicit Methods

State of the art finite element and finite difference methods use explicit solvers to provide stability and performance. Implicit methods depend on global solves of sparse matrices. Liszt has added language-level support for solving sparse matrices, and integrates the PETSc solver as a backend.

Sparse matrices are tied to the topology of the mesh, allowing for simple referencing. Implicit formulations of finite difference methods have a regular matrix structure, currently supported by Liszt. Higher order finite element methods require multiple, different submatrices per element in its matrix formulation, currently in development.

The implicit version of Joe has been ported to Liszt, reducing its codebase from 3106 lines to 1520 lines (this disregards the 20 000+ lines of MPI boilerplate code in C++ Joe). MPI performance is comparable for both the explicit and implicit versions of Joe.

### GPGPU Runtime

The GPU backend implements gathering and reductions in native NVidia C, and manages mesh and field data on the GPU. The JIT phase for the GPU performs transformations to convert standard scatter-based operations into gathers, allowing arbitrary code to be executed on the GPU.

### Current and Future Work

We are currently working on:
- DSL advances through Polymorphic Embedding
- GPGPU-specific loop transformations
- FEM & DG support through canonical elements

Future work:
- Release private beta at upcoming Codeathon
- Uncertainty quantification support
- Transformations between scatters, gathers & reduces
- A hybrid runtime combining MPI and GPGPU

---

**Overview**

Liszt! A DSL for Mesh-Based PDEs