FusionSC – Simplifying the daily work in fusion device modeling

A. Knieps[1], D. Böckenhoff[2], D. Boeyaert[3], S. Bozhenkov[2], M. Endler[2], Y. Feng[2], J. Geiger[2], Y. Liang[1], L. Liao[4], Y. Suzuki[5], and the W7-X Team

[1] *Forschungszentrum Jülich GmbH, Jülich, Germany*

[2] *Max-Planck-Institut für Plasmaphysik, Greifswald, Germany*

[3] *University of Wisconsin-Madison, 53706 Madison (Wisconsin), USA*

[4] *Institute of Plasma Physics, Chinese Academy of Sciences, Hefei 230031, China*

[5] *Graduate School of Advanced Science and Engineering, Hiroshima University, 739-8527 Higashi-Hiroshima, Japan*

The FusionSC (short for Fusion Scientific Computation) library is a support library designed to assist fusion researchers with common computation tasks in magnetic confinement fusion - particularly in the handling of magnetic fields and machine geometries, and simpler calculations related to these, such as Poincare plots.

The goal of this project was to bundle the functionalities of various webservices currently employed at Wendelstein 7-X (Coil & Component Database, Field Line Tracer, Magnetic Field Calculator, Mesh Server) into a self-contained system, so that it could then be deployed outside and be extended more easily with new features (such as fully diffusive tracing, heat load imaging, and other magnetic field types).

Current capabilities

Presently, FusionSC ships with the following features:

- Field line tracing: FusionSC's field line tracer can be employed for regular, conective-diffusive and fully-diffuse field line tracing. It tracks geometry-intersections, can calculate Poincaré maps (with connection-length information), can calculate and employ EMC3-style reversible field line mappings [3], and supports adaptive error-estimation based step size control.

- Fourier surface calculation: Using the field line tracer, FusionSC can calculate Fourier representations of magnetic surfaces. Through an extended multi-turn coordinate system, it can also calculate Fourier coefficients for islands on non-integer rational surfaces (such as 5/6 islands).

- Magnetic field calculation: The magnetic field calculation subsystem can calculate magnetic fields on grids, surfaces (including Fourier expansions of the field on said surfaces),

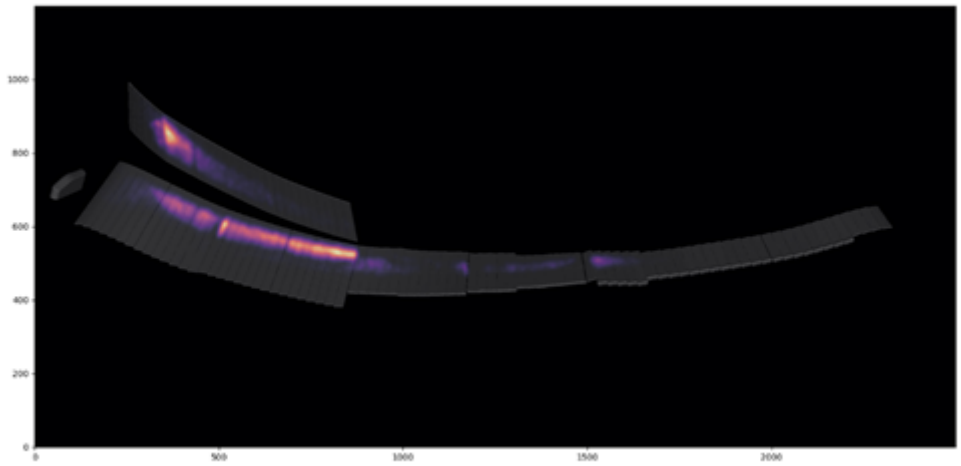

Figure 1: Heat load distribution on the W7-X divertor in magnetic standard configuration calculated using diffusive tracing

and arbitrary points. Supported fields currently are 2D poloidal flux distributions (for 2D equilibria), Biot-savart filament fields, Magnetized spheres (regularized dipoles), and externally precomputed fields, as well as translations and rotations of said fields.

- Geometry handling: The geometry handling subsystem can scale, translate, rotate, and merge meshes, toroidally extrude 2D R-Z meshes into 3D space, and can read and write many standard mesh formats (such as PLY or STL).

- Heat load calculation: FusionSC supports the calculation of heat loads from impact point clouds (as they arise in Monte-Carlo simulations) through a virtual camera based approach [2] capable of handling arbitrary mesh geometries (see figure 1 for an example).

- A server infrastructure offering warehouse servers, compute servers, and a special type of frontend node that can act both as a load-balancer and an interconnect (thus relieving the client-server connection of the need to pass data between server-side nodes)

Architecture

Integrated client-server model

Similar to a typical webservice, the FusionSC library employs a client-server architecture. However, the library bundles both the client- and server-side in a single codebase. In the default setup, requests are handled by an implicitly started in-process backend. Using a remote calculation server only requires a single operation by the user to switch to the remote backend.

Unlike typical webservice architectures, FusionSC does not use a stateless request-response model with strict client-server roles. Instead, it relies on a symmetric stateful object-oriented RPC system [4], which bidirectionally multiplexes all data- and computation-requests over a single connection. In this architecture, the server side can also make requests to the client, allowing the client to transfer data on-demand and to also act as a connection hub between different calculation servers (which is e.g. useful when the server running the calculation can not directly connect to the database hosting component data).

Data model

For its calculations, FusionSC heavily encourages the use of declarative high-level inputs. Fields and geometries are specified as a tree of operations, which are interpreted by the server to obtain the requested information.

However, besides the usual physics specifications (such as Biot-Savart nodes or individual geometry meshes), there are 2 very important data kinds supported:

- Remote references: FusionSC supports data types to refernce externally stored data. These references are not declarative, but functional (by pointing to an RPC service object). These handles are passed by reference - even across remote calls - and are only downloaded by the consumer when absolutely neccessary (avoiding redundant copies and network round-trips through the client).

- Abstract named nodes: These serves as high-level machine-specific specifications for geometries and fields (and are often machine-dependent) and can not be directly interpreted by the server. Instead, before submitting these inputs to the backend, the client will consult a series of resolvers to transform the declarative information within into concrete fields, geometries, and coil filaments. FusionSC comes pre-equipped with W7-X and J-TEXT-specific resolvers, and a general lookup-based resolver infrastructure (see below).

In combination, these two data types are used to transparently implement component-database functionality: At startup, various storage sources are queried for stored components and coil geometries. These locally cached tables are then consulted by the resolution system whenever a component name needs to be resolved. The result is then passed - usually by reference - to the backend, which will request the full data from the original source on demand.

Data storage

FusionSC provides 3 ways for managing data storage:

- For simple write-once storage of individual objects, a special *archive* file format is provided. An archive contains a single data object plus the tree of all referenced remote objects. This format is optimized for speed - archives are mapped into process memory, allowing the operating system to demand-load sections of the file once the contained data are actually accessed.

- For more complex storage needs, FusionSC also provides an object database (dubbed *warehouse*). This database contains at its top level a mutable file-system-like folder structure, in which users can deposit individual data objects. Warehouses are designed for multi-user applications, and support remote access, crash safety, multi-process access, data compression, and backups.

- In case data need to be processed outside Python, FusionSC also comes with a rich serialization library, which can read and write JSON, YAML, CBOR, MSGPACK, UBJSON, and BSON.

Similar to the avoidance of duplicate data transfer, FusionSC's native formats are also designed to avoid duplicate storage - if the same object is used twice in different places, the underlying storage is aliased. Users therefore do not need to spend much effort on efficient data organization, and can instead freely save redundant data (e.g. by attaching the vacuum field to every equilibrium calculation).

The data engine is designed around Cap'n'proto's [4] native data format, but also includes a compatibility layer to handle typical scientific Python payloads, such as dicts, lists, and NumPy arrays.

Acknowledgements

This work has been carried out within the framework of the EUROfusion Consortium, funded by the European Union via the Euratom Research and Training Programme (Grant Agreement No 101052200 — EUROfusion). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

References

[1] S. Bozhenkov et al, Fus. Eng. Des. **88** (2013)

[2] A. Knieps et al, Plasm. Phys. Control. Fus. **64** (2022)

[3] Y. Feng et al, Phys. Plasmas **12** (2005)

[4] K. Verda, The Cap'n'Proto RPC system, https://capnproto.org