Loki: Freely Programmable Source-to-Source Translation for IFS and beyond

Balthasar Reuter, Ahmad Nawab, Michael Staneker,
Johan Ericsson, Michael Lange

Research Department, ECMWF, Bonn (Germany) {firstname}.{lastname}@ecmwf.int

Motivation

- **Perfomance portability** of Numerical Weather Prediction (NWP) codes across a broad range of HPC architectures, including **accelerators** (such as GPUs), from a **single code base**
- **Static code analysis/linting** of source code to aid development

Challenges

- Different **programming paradigms** and environments
- **Hardware-specific** optimisation (loop order, memory layout, ...)
- Handling a **large** and **complex Fortran code base**
- **Compatibility** with operational requirements and scientific changes

Methodology

Source-to-source (S2S) translation tool to inspect/transform code:

- **Static code analysis** using internal representation
- **Build-time transformation** of source code using bespoke recipes

Open development on Github

Source code & bug tracker

/ecmwf-ifs/loki

Documentation

sites.ecmwf.int/docs/loki

Jupyter Notebook Tutorials

/ecmwf-ifs/loki/tree/main/example

Loki: overview and internal representation

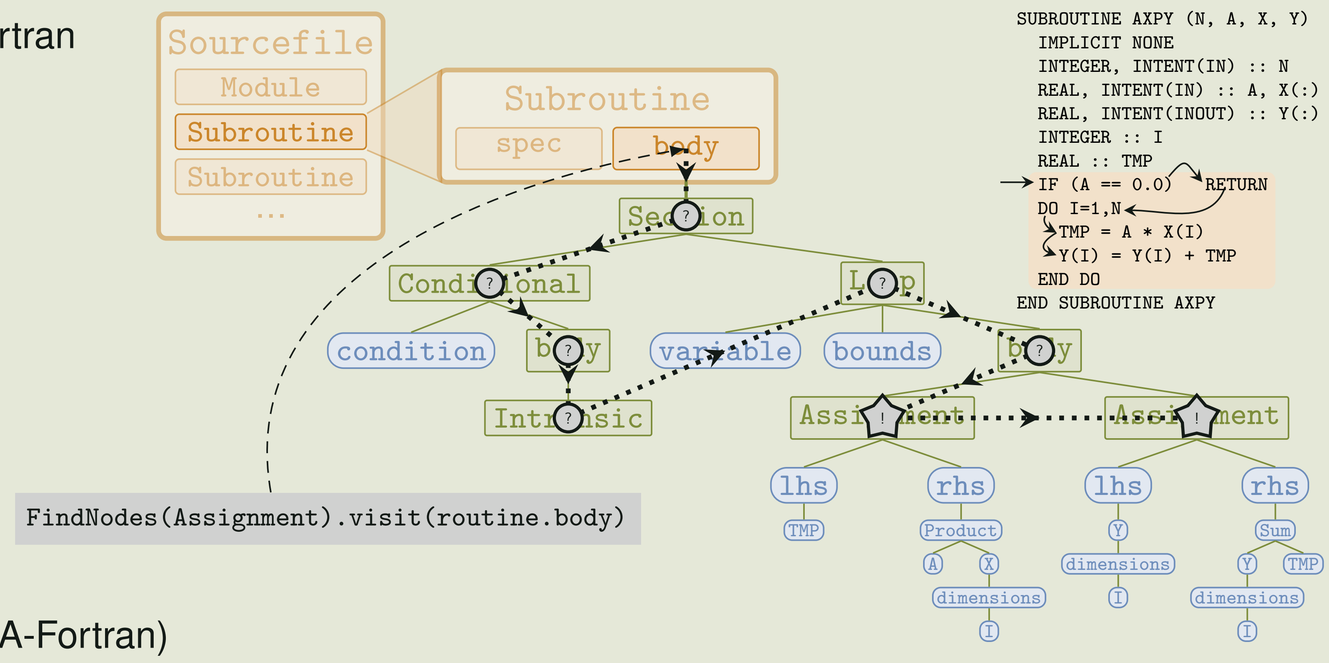
Loki is a **Python package** to encode S2S translation recipes for Fortran

Core library: Internal representation (IR) and API to **encode custom transformations** or **analysis/linting pipelines**

- *Fparser2*[1] is used to generate parse tree of Fortran source
- The parse tree is converted into Loki's **two-level IR**, separating (Fortran-tinted) control-flow from expression tree

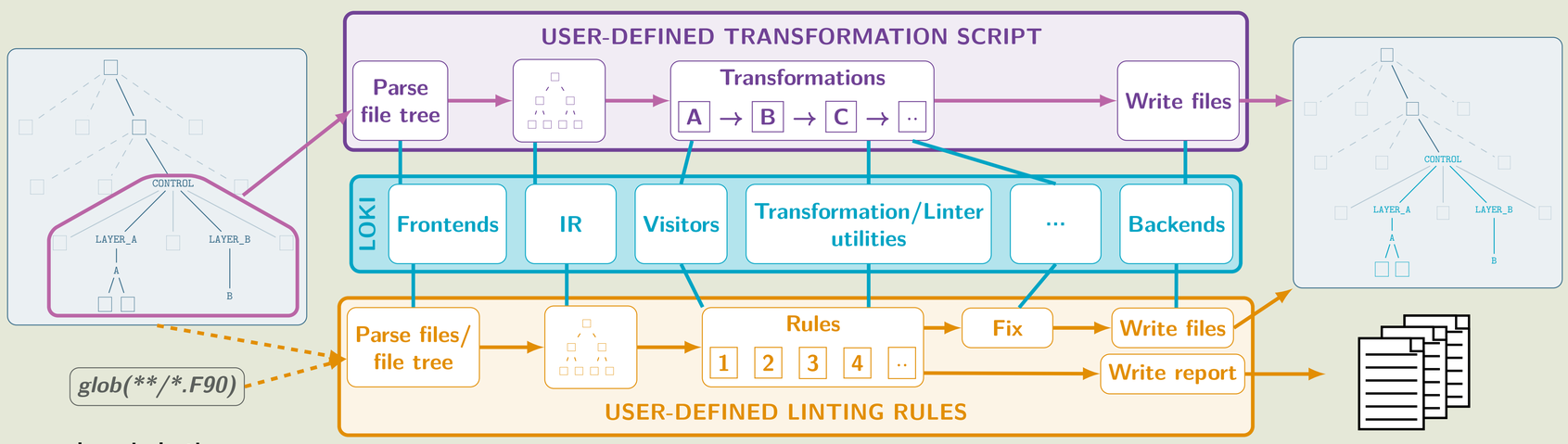
Features:

- **Visitors** are used to traverse and transform the IR
- Scope-aware symbol tables **manage type information**
- **Scheduler** builds a dependency graph for call trees across source files and allows for **inter-procedural analysis**
- **Backends** to generate Fortran (experimental: C, Python, or CUDA-Fortran)



Bulk transformation and analysis of source code

- Typical S2S translation recipes consist of **multiple bespoke transformation steps**
- User-defined pipeline of transformation steps can be built using core library utilities
- **Scheduler** applies transformations in the order of the dependency graph
- **CMake integration** automatically updates dependencies of build system targets
- Same infrastructure unlocks custom static code analysis and experimental fixing of coding rule violations



Architecture-specific recipes to generate bespoke optimisations

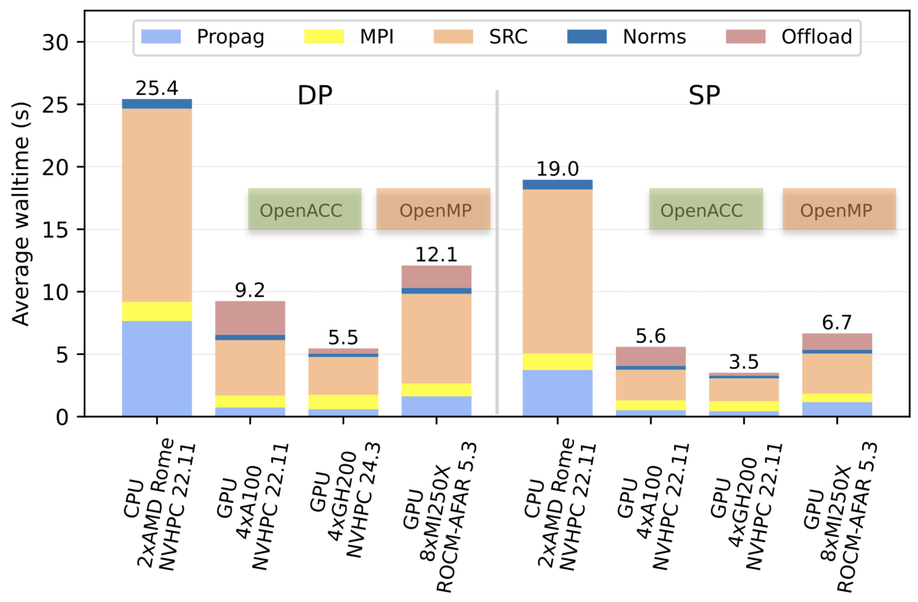
IFS model components with open-source mini-apps serve as **proxies for full-model algorithms** to develop transformation recipes

/ecmwf-ifs/ecwam

ecWAM is the **operational IFS wave model**, consisting of wave propagation and physics

GPU adaptation recipes are composed from multiple transformations, e.g., swapping horizontal/vertical loop for additional parallelism, inlining, or handling of temporaries

Multiple programming model backends improve portability across vendors, e.g., OpenACC and OpenMP for NVIDIA and AMD GPUs



	Hardware	Propag	MPI	SRC	Norms	Offload
DP	CPU	7.66	1.53	15.46	0.77	0.00
	GPU A100	0.73	0.95	4.44	0.43	2.69
	GPU 4xH100	0.59	1.15	3.02	0.28	0.41
	GPU MI250X	1.63	1.03	7.17	0.48	1.79
SP	CPU	3.73	1.34	13.12	0.78	0.00
	GPU A100	0.53	0.78	2.45	0.32	1.51
	GPU H100	0.44	0.80	1.83	0.22	0.22
	GPU MI250X	1.17	0.67	3.21	0.32	1.30

*Average walltime (s)

Comparison of average wall time in double (DP) and single precision (SP) for ecWAM execution on (a) a single node of ECMWF's Atos HPCF, (b) a single node of ECMWF's AC GPU nodes (4x NVIDIA A100 GPUs), (c) a single node of EuroHPC's JUPITER supercomputer (4x NVIDIA GH200), and (d) a single node of the EuroHPC LUMI-G supercomputer (8x AMD MI250X)

The work presented in this poster has been produced in the context of the European Union's Destination Earth Initiative and relates to tasks entrusted by the European Union to the European Centre for Medium-Range Weather Forecasts implementing part of this Initiative with funding by the European Union.