Loki: A Source-to-Source Translation Tool for Numerical Weather Prediction Codes

Michael Lange[1], Michael Staneker[1], Ahmad Nawab[1], Balthasar Reuter[1], Olivier Marsden[2]

[1]ECMWF, Bonn (Germany) [2]ECMWF, Reading (UK) {firstname}.{lastname}@ecmwf.int

Motivation

- **Performance portability** of Numerical Weather Prediction (NWP) codes on **heterogeneous HPC** (CPUs, NVIDIA/AMD GPUs)
- **Single code base**: use **existing Fortran as-is**
- **Automated, deterministic** changes to **many files** at **build time**

Challenges

- Different **programming models** (OpenACC, OpenMP, CUDA)
- **Hardware-specific** optimisation (loop order, memory layout, ...)
- **Separate GPU memory spaces** require explicit data offload
- **Large**, **evolving Fortran code base**

Methodology

Source-to-source (S2S) translation tool for IFS code:

- **Build-time transformation** using bespoke, **expert-guided** recipes encoded via an **agent-friendly Python API**
- Same internal representation enables static analysis and linting

Agent-assisted recipe development

Agent drafts recipe → **Loki API + tests** validate → **Scheduler** applies it at build time, **deterministically**

For **bespoke backends** and **bespoke Fortran coding patterns**

Loki: overview and internal representation

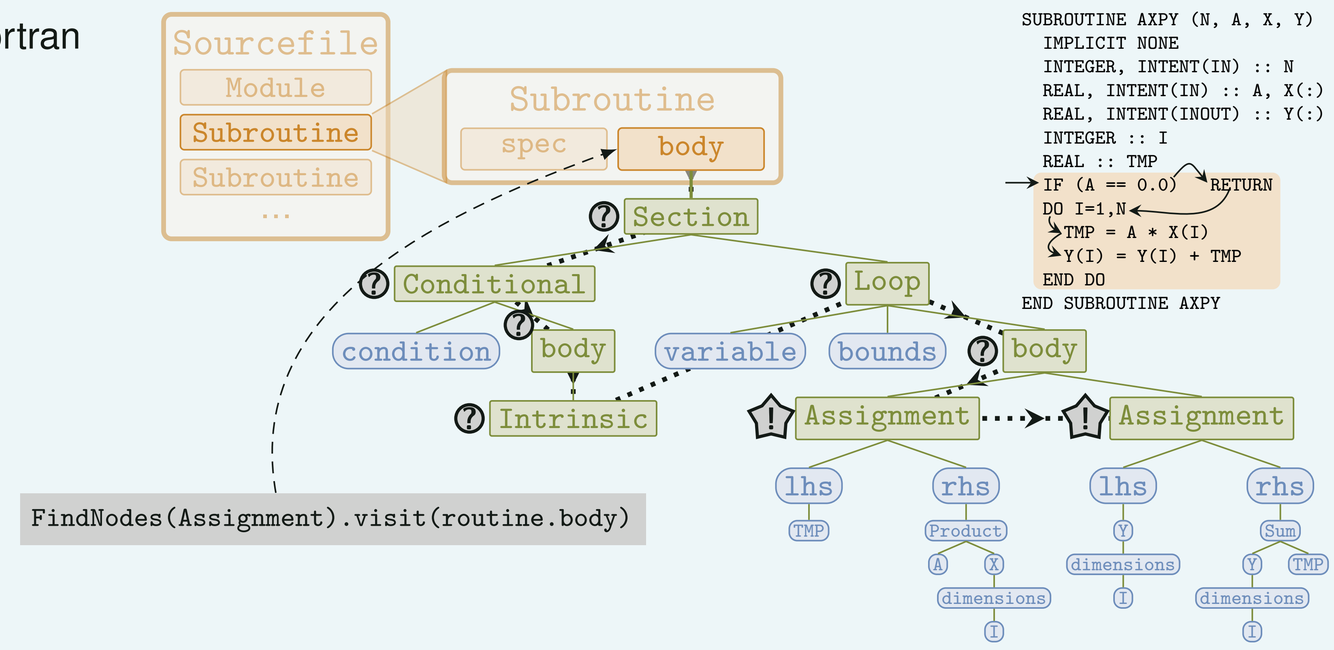
Loki is a **Python package** to encode S2S translation recipes for Fortran

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

- **Frontends**: *fparser2*, OMNI, fast REGEX pre-parser
- Loki's **two-level IR** separates (Fortran-tinted) control-flow from expression tree

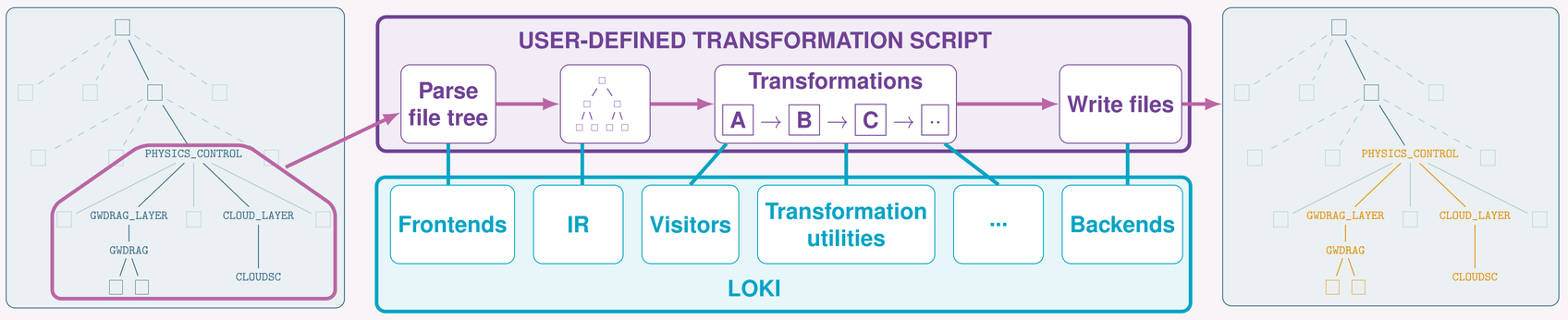
Features:

- **Visitors** traverse/transform IR; scoped **type information**
- **Scheduler**: call-tree graph for **inter-procedural** transformations
- **Backends**: Fortran, C/C++, CUDA/HIP, CUDA Fortran, Python
- **Dataflow analysis**, **CLI** and **CMake integration**



Bulk transformation of source code

- User-defined pipelines of **bespoke steps**, built from core library utilities
- **Scheduler** applies steps along the dependency graph
- **CMake integration** at build time
- Also drives static analysis and linting



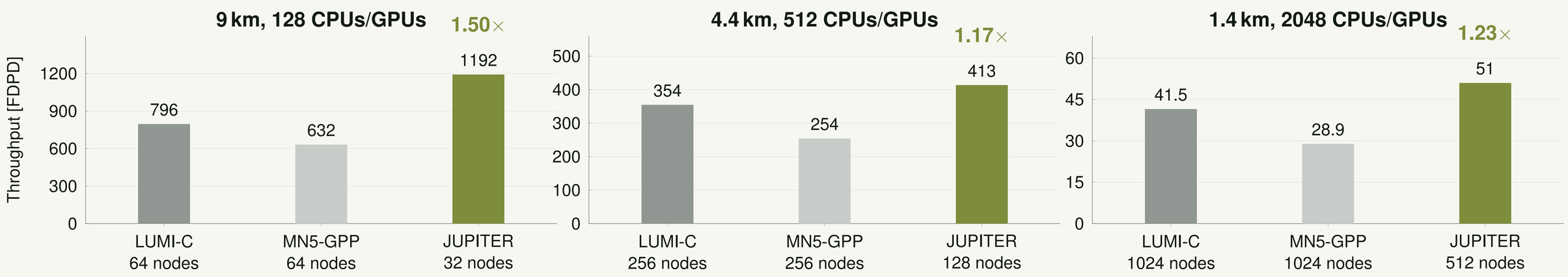
GPU adaptation of the IFS

Transformation recipes composed per target:

- **SCC** (single-column coalesced): loop swap, demotion
- **Temporaries**: hoisting, pool and stack allocators
- **Block-index injection**, driver-loop handling
- **OpenACC**, **OpenMP offload**, **CUDA/HIP**

Automated data offload to GPU memory:

- **Dataflow analysis** of call tree finds access of derived-type members
- Generated **deepcopy** offload and pull-back
- **FIELD API** offload, **blocked** and **asynchronous**
- Offload of **module (global) variables**



IFS atmosphere *throughput in forecast days per day (FDPD) at three resolutions on EuroHPC CPU systems vs.* ***GPUs on JUPITER*** *(4x GH200 per node). On half the nodes, JUPITER achieves* ***1.2–1.5× higher throughput****;* ***Loki generates a large part of the GPU code*** *(grid-point computations).*

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.

Source & issues
github.com/ecmwf-ifs/loki

Documentation
sites.ecmwf.int/docs/loki

Tutorials
github.com/ecmwf-ifs/loki/tree/main/example