Automating GPU adaptation of NWP single column physics using Loki

A. Nawab M. Lange B. Reuter M. Staneker

ahmad.nawab@ecmwf.int

European Centre for Medium-Range Weather Forecasts

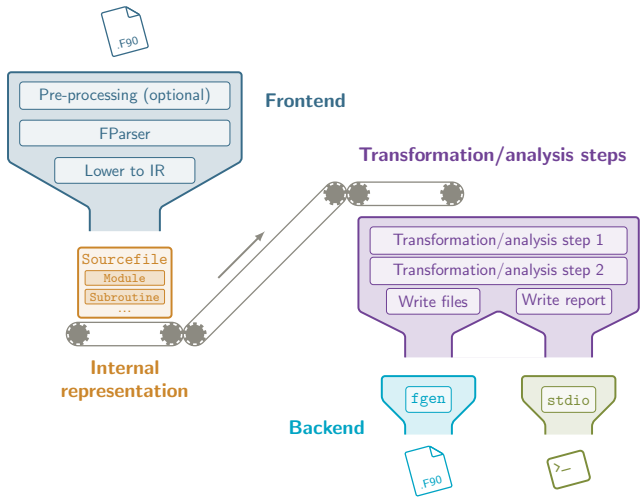
ECMWF

Introduction

- Michael S. just gave us an in-depth explanation of how NWP single column algorithms can be adapted to optimally exploit GPU hardware
- The current talk focuses on Loki, our source-to-source translation tool used to automate this adaptation

Loki - source-to-source translation tool

- **Loki** - programmable source-to-source translation toolchain for Fortran codes
- Written in Python
- Typical Loki pipeline:



Loki - parsing

- First parsing step: use *Fparser*[1] to obtain an abstract syntax tree (AST)
  - Supports all of F2003, F2008 features added regularly
- AST enriched with context and metadata to form a custom internal representation (IR)

Parse expression-tree

```
>>> from loki import parse_fparser_expression, Scope
>>> expr = parse_fparser_expression('a*(b*(c+(d+e)))',
...                                 scope=Scope())
>>> print(expr)
a*(b*(c + (d + e)))
>>> type(expr)
<class 'loki.expression.symbols.Product'>
```

Parse sourcefile

```
>>> from loki import Sourcefile
>>> source = Sourcefile.from_file('src/phys_mod.F90',
...                               preprocess=True)
>>> type(source)
<class 'loki.sourcefile.Sourcefile'>
>>> len(source.modules)
1
>>> len(source.subroutines)
0
```

Parse subroutine

```
>>> from loki import Subroutine
>>> fcode = """subroutine add_ints(a,b,c)
... implicit none
... integer, intent(in) :: a,b
... integer, intent(out) :: c
...
... c = a+b
... end subroutine add_ints"""

>>> routine = Subroutine.from_source(fcode)
>>> type(routine)
<class 'loki.subroutine.Subroutine'>
>>> print(routine.name)
'add_ints'
>>> len(routine.arguments)
3
```

Loki - IR

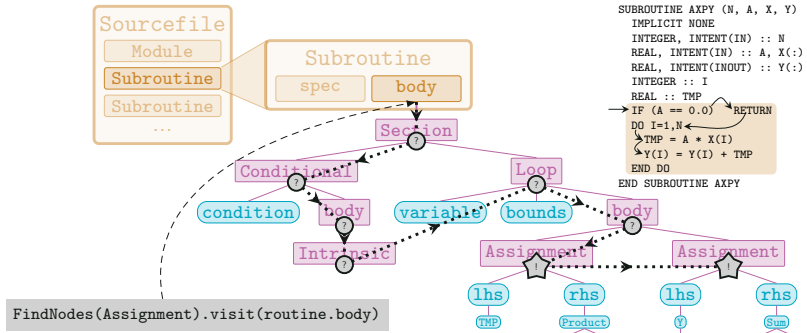
- Fortran language constructs are stored as individual IR nodes

```
### extract module from src/phys_mod.F90
>>> phys_mod = source['phys_mod']
>>> phys_mod
Module:: phys_mod
>>> from loki import fgen
>>> print(fgen(phys_mod.spec))
USE iso_fortran_env

USE omp_lib

IMPLICIT NONE

INTEGER, PARAMETER :: sp = REAL32
INTEGER, PARAMETER :: dp = REAL64



INTEGER, PARAMETER :: lp = dp!! lp : "local" precision


INTEGER, PARAMETER :: ip = INT64

REAL(KIND=lp) :: cst1 = 2.5, cst2 = 3.14
INTEGER, PARAMETER :: nspecies = 5
```

```
>>> phys_mod.spec.view()
<Section::>
  <Import:: iso_fortran_env => ()>
  <Comment:: >
  <Import:: omp_lib => ()>
  <Comment:: >
  <Intrinsic:: IMPLICIT NONE>
  <Comment:: >
  <VariableDeclaration:: sp>
  <VariableDeclaration:: dp>
  <CommentBlock:: >
  <VariableDeclaration:: lp>
  <CommentBlock:: >
  <VariableDeclaration:: ip>
  <Comment:: >
  <VariableDeclaration:: cst1, cst2>
  <VariableDeclaration:: nspecies>
```

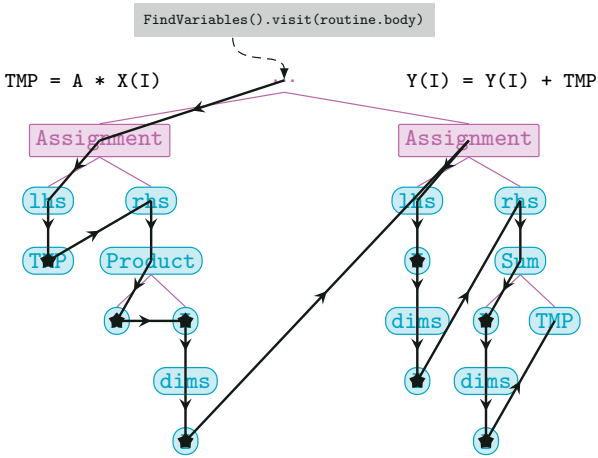
Loki - IR

- The Loki IR consists of two levels:
  - Outer level corresponds to control-flow nodes, e.g. `Assignment`, `Loop` (purple nodes)
  - Inner level corresponds to expression nodes, e.g. `Scalar`, `Array`, `Product` (blue nodes)
    - Built on top of the *Pymbolic*[2] library



Loki - visitors

- The previous slide showed an example of the `FindNodes` visitor utility to lookup control-flow nodes
- Loki relies heavily on visitor utilities to traverse/modify the IR
- Analogous utilities also exist for expression nodes, e.g. `FindVariables`



Loki - adding new functionality

- Extend functionality of `FindNodes` to create `FindNodesDepth` visitor
  - Perform node lookup *and* retrieve depth of node in IR

```python
from loki import FindNodes

class FindNodesDepth(FindNodes):

    def __init__(self, match, greedy=False):
        super().__init__(match, mode='type', greedy=greedy)


    # copied from FindNodes.visit_Node
    def visit_Node(self, o, **kwargs):
        ret = kwargs.pop('ret', self.default_retval())
        depth = kwargs.pop('depth', 0)
        if self.rule(self.match, o):
            ret.append((o, depth))
            if self.greedy:
                return ret
        for i in o.children:
            ret = self.visit(i, depth=depth+1, ret=ret, **kwargs)
        return ret or self.default_retval()
```

- `FindNodesDepth` can be created by changing only 4 lines of `FindNodes`

Loki - adding new functionality

- Apply both visitors to `SUBROUTINE AXPY` from earlier example

`FindNodes`

```
>>> from loki import Assignment, Subroutine, FindNodes
>>> routine = Subroutine.from_source(...)
>>> assigns = FindNodes(Assignment).visit(routine.body)
>>> len(assigns)
2
>>> print(assigns[0])
Assignment:: tmp = a*x(i)
```

`FindNodesDepth`

```
>>> from loki import Assignment, Subroutine, FindNodesDepth
>>> routine = Subroutine.from_source(...)
>>> assigns = FindNodesDepth(Assignment).visit(routine.body)
>>> len(assigns)
2
>>> print(assigns[0])
(Assignment:: tmp = a*x(i), 2)
```

Loki - visitors to modify the IR

Transformer

- Traverses the IR and replaces nodes according to a given map
- Supports 1-to-many mappings (replace node with sequence)

SubstituteExpressions

- Traverses the IR and recurses into every expression tree
- Replaces expression tree nodes according to a given map

```
def remove_loops(routine, dimension):
    """
    Replace vector loops with their own bodies
    """
    loop_map = {}
    for loop in FindNodes(Loop).visit(routine.body):
        if loop.variable == dimension:
            loop_map[loop] = loop.body

    routine.body = Transformer(loop_map).visit(routine.body)
```

Loki - transformations

- Source code transformations can be defined using the `Transformation` class
  - `Transformation` class is the **building-block** in a transformation pipeline

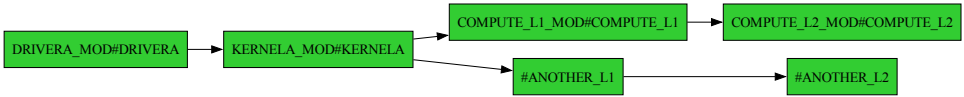
- Entry points at multiple levels:
  `transform_subroutine(), transform_module(), transform_file()`
- Can be applied to container data structures (`Sourcefile, Module, Subroutine`):
  ```
  MyTransformation().apply(sourcefile)
  MyTransformation().apply(module)
  MyTransformation().apply(kernel)
  ```
- `apply()` automatically dispatches to the relevant entry point level

Loki - bulk processing source trees

[WARNING] Scheduler is undergoing an incremental re-write!

- `Scheduler` applies transformations in bulk over call-trees

```
# Dependency tree for projA:
# projA: driverA -> kernelA -> compute_l1 -> compute_l2
#                        |
#                        | --> another_l1 -> another_l2
...
### define scheduler config via dictionary or config file
>>> config = {...}
>>> from loki import Scheduler
>>> scheduler = Scheduler(paths='projA', includes='projA/include', config=config, seed_routines=['driverA'])
>>> scheduler.callgraph('callgraph_simple')
```



- `Scheduler.apply(transformation)` applies a transformation to all items in the dependency graph
  - Can be applied "forward" or "backward"

Loki - SCC: Single Column Coalesced

- SCC "family" of transformations used to port IFS physics to GPU

- The baseline SCC is a multi-step transformation pipeline:
  - Resolve associates, explicit vector notation, masked conditionals, etc.
  - Remove CPU-style vector loops over horizontal
  - Demote array variables where possible
  - Re-insert horizontal vector loops around kernel
  - Insert OpenACC / OpenMP pragmas as appropriate

- Further (optional) transformations to reduce temporary array allocations on device:
  - Loki-SCC-hoist: Hoist temporaries to drivers
  - Loki-SCC-stack: Inject pool allocator with pre-computed stack size

- SCC transformations rely on certain coding norms
  - Remember, Loki is not a DSL ;)

Loki - SCC: single column formatting rules

- SC1: horizontal indexing
  - Variables referring to horizontal indices should be named consistently e.g. `JL`
- SC2: horizontal looping
  - All loops over the innermost, horizontal array dimension should be explicit (explicit array notation also ok)
- SC3: function calls from inside horizontal loops
  - Inside tight horizontal loops of type `DO JL=KIDIA,KFDIA` calls should be restricted to intrinsics
- SC4: no horizontal indirection
  - Indirect addressing on the innermost, horizontal array index shall not be used.
- SC5: no horizontal reduction across vector loop
  - No reduction operations (reducing the elements of an array into a single value) across the vector loop index `JL`
  - Although this introduces a horizontal dependency and is thus strictly speaking a violation of single column rules, Loki has functionality to facilitate this if needed
- SC6: no horizontal index array accessing with an offset
  - Arrays accessed in the horizontal dimension via the horizontal index should be accessed without any offset like e.g. `JL + n`

Loki - **loki-lint.py**

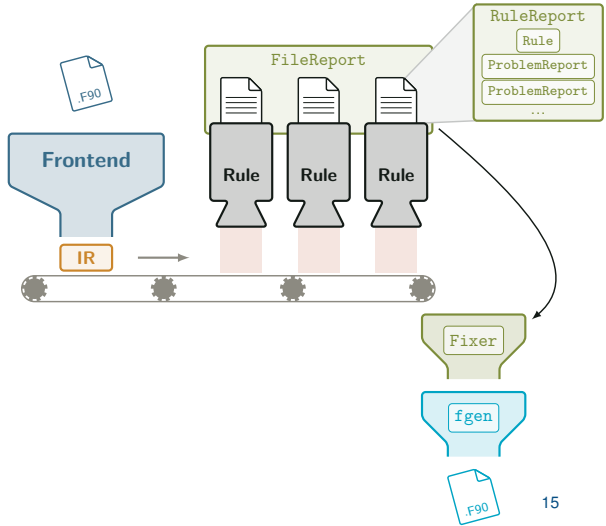
loki-lint.py: CLI tool to check code-base or call-tree against a configurable set of rules

CI code checking/linting

- **loki-lint.py** is run as part of the IFS CI suite

Interprocedural debugging utilities

- Argument intent linter: verify the correctness of subroutine dummy argument intent
- Argument size mismatch linter: check for size mismatch errors between arguments and dummy arguments



Loki - open development on github

- **Source code repository** on github: `https://github.com/ecmwf-ifs/loki`
- **Documentation** and install instructions: `https://sites.ecmwf.int/docs/loki/`
- **Bug tracking** and feature wish list: `https://github.com/ecmwf-ifs/loki/issues`
- **Tutorial** Jupyter notebooks: `https://github.com/ecmwf-ifs/loki/tree/main/example`
- **CLOUDSC** with multiple Loki transformations: `https://github.com/ecmwf-ifs/dwarf-p-cloudsc`

Thank you! Any questions?

References

1. Science and Technology Facilities Council. *fparser.* https://github.com/stfc/fparser.
2. Klöckner, A. & Contributors. *Pymbolic.* https://github.com/inducer/pymbolic.

The work presented in this paper 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. Views and opinions expressed are 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.