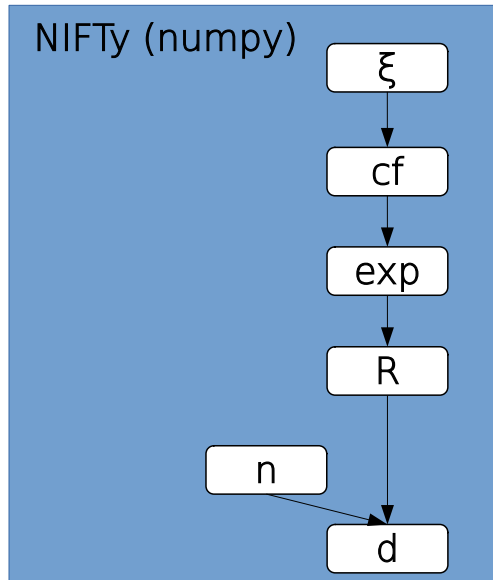


Extending NIFTy and Jax

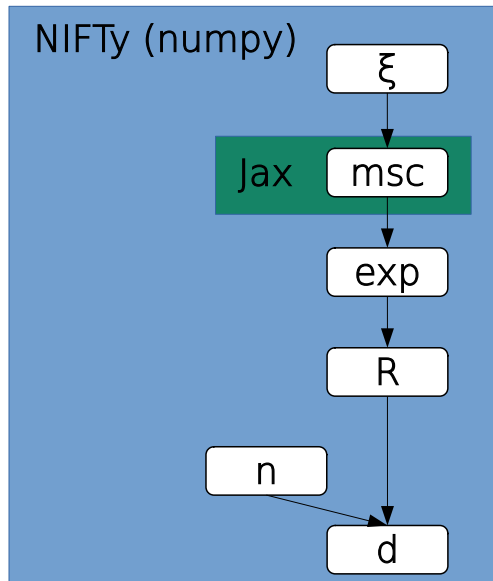
Jakob Roth, MPA Garching

November 21, 2023

NIFTy Model



NIFTy Model



NIFTy/ NIFTy.Re Model

Martin Reinecke · ducc

ducc Project ID: 3606 ☆ Star 0

2,455 Commits 26 Branches 39 Tags 591.6 MiB Project Storage

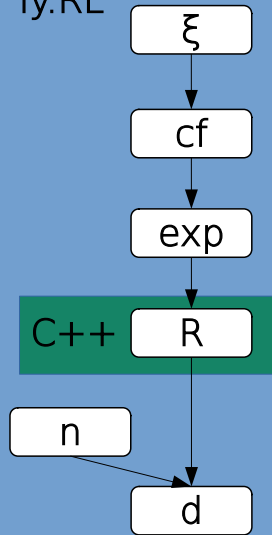
release GIL in a few more places
Martin Reinecke authored 1 week ago e673cb5e

ducc8 ducc History Find file Clone

README Affero General Public License v1.0 CHANGELOG

Name	Last commit	Last update
.github/workflows	adjust (hopefully) Julia and Rust bindings	4 months ago
doc	update docs	1 year ago
fortran	tweak	1 month ago
julia	adjust (hopefully) Julia and Rust bindings	4 months ago
not_yet_integrated	make IPS4D work with ducc threads	5 months ago
python	release GIL in a few more places	1 week ago
rust	adjust (hopefully) Julia and Rust bindings	4 months ago
src	pull improvements from vector_wigner branch	1 week ago
.gitignore	Add basic Rust stuff	9 months ago
.gitlab-ci.yml	try to fix CI	3 months ago
ChangeLog	wgridder beyond-horizon fixes	1 week ago
Dockerfile	try to adjust to CI changes	5 months ago
LICENSE	cleanup	3 years ago
MANIFEST.in	add Python interface for Wigner 3j code	2 months ago
README.md	require Python >=3.8	3 months ago
compile_hipsycl_nvidia	sync and cleanup	1 year ago
pyproject.toml	require Python >=3.8	3 months ago
release_instructions.md	First draft for release process	6 months ago
setup.cfg	require Python >=3.8	3 months ago
setup.py	bump version number	1 month ago

NIFTy/ NIFTy.RE



Interfaces

- Jax to NIFTy
- C++ to NIFTy
- C++ to NIFTy.RE
- Tensorflow in Jax

Jax to NIFTy

```
1 class JaxOperator(Operator):
2     """Wrap a jax function as nifty operator.
3
4     Parameters
5     -----
6     domain : DomainTuple or MultiDomain
7         Domain of the operator.
8
9     target : DomainTuple or MultiDomain
10        Target of the operator.
11
12    func : callable
13        The jax function that is evaluated by
14        the operator.
15    """
16    def __init__(self, domain, target, func):
```

Why is it that easy?

- NIFTy and Jax both in Python
- Jax as built in AD

→ JaxOperator connects Jax AD with NIFTy AD.

Jax to NIFTy

NIFTy

```
1 y = op(x)
2
3 # for linear op
4 y = op.adjoint(x)
5
6 # for non-linear op
7 jac = op(lin).jac
8 dy = jac(dx)
9 dx = jac.adjoint(dy)
```

Jax

```
1 y = op(x)
2
3 # for linear op
4 x = linear_transpose(op, x)(y)[0]
5
6 # for non-linear op
7 y, dy = jvp(op, [x], [dx])
8 y, op_vjp = vjp(op, x)
9 dx = op_vjp(dy)
```

Jax to NIFTy

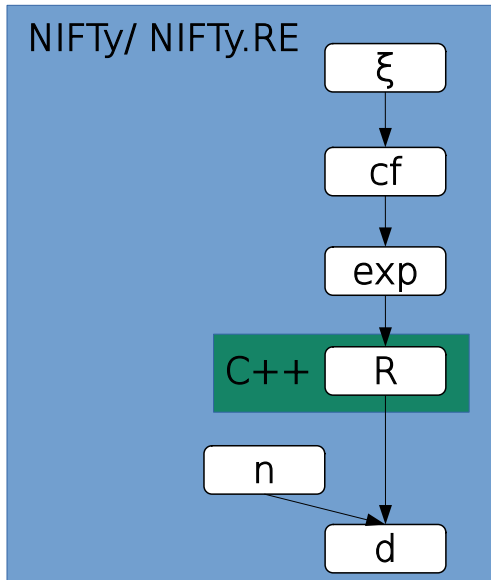
```
1 class JaxLinearOperator(LinearOperator):
2     def __init__(self, domain, target, func):
3         self._domain = makeDomain(domain)
4         self._target = makeDomain(target)
5         self._func = func
6         self._func_T = jax.linear_transpose(func, inp)
7         self._capability = self.TIMES | self.ADJOINT_TIMES
8
9     def apply(self, x, mode):
10        if mode == self.TIMES:
11            fx = self._func(x.val)
12            return makeField(self._target, fx)
13        fx = self._func_T(x.val)
14        return makeField(self._domain, fx)
```


Jax to NIFTy

```
1 class JaxOperator(Operator):
2     def __init__(self, domain, target, func):
3         ...
4
5     def apply(self, x):
6         if is_linearization(x):
7             res = self._func(x.val.val)
8             bwd = partial(self._bwd, x.val.val)
9             fwd = partial(self._fwd, x.val.val)
0             jac = JaxLinearOperator(domain, target, fwd, bwd)
1             return x.new(makeField(target, res), jac)
2
3     res = self._func(x.val)
4     return makeField(target, res)
```

C++ to NIFTy

- Bind C++ functions to python: e.g. with pybind11
- No AD in C++: Additionally code Jacobian and Jacobian adjoint



C++ to NIFTy: Summary

C++:

- Apply Operator
- Apply Jacobian
- Apply adjoint Jacobian

pybind11:

- Expose C++ functions to python

NIFTy:

- Interface to NIFTy
- Wrap C++ computations as a nifty operator

Examples: $y = x^2$

```
1 class SquareOperator {
2     public:
3         SquareOperator() {}
4         py::array apply(const py::array &inp) const {
5             auto pyout = inp * inp;
6             return pyout;
7         }
8
9         Linearization apply_with_jac(const py::array &d) {
10            // Implementation of vjp and vjp
11
12            return Linearization(apply(d), jvp, vjp);
13        }
14    };
```

pybind11

- "pybind11 is a lightweight header-only library that exposes C++ types in Python and vice versa, mainly to create Python bindings of existing C++ code."



<https://github.com/pybind/pybind11>

```

1 #include <pybind11/numpy.h>
2 #include <pybind11/pybind11.h>
3
4 // Implementation of SquareOperator and Linearization
5 // ...
6
7 PYBIND11_MODULE(cpp_for_nifty, m) {
8     py::class_<SquareOperator>(m, "SquareOperator")
9         .def(py::init<>())
10        .def("apply", &SquareOperator::apply)
11        .def("apply_with_jac", &SquareOperator::apply_with_jac);
12
13    py::class_<Linearization>(m, "Linearization")
14        .def(py::init<const py::array &, function<py::array(const py::array
15        &>>,
16                function<py::array(const py::array &>>>())
17        .def("position", &Linearization::position)
18        .def("jac_times", &Linearization::jac_times)
19        .def("jac_adjoint_times", &Linearization::jac_adjoint_times);
20 }

```

C++ to NIFTy

```
1 import numpy as np
2 import nifty8 as ift
3
4
5 class Pybind11Operator(ift.Operator):
6     def __init__(self, .., pb11_op):
7         ...
8         self._op = pb11_op()
9
10    def apply(self, x):
11        if ift.is_linearization(x):
12            lin = self._op.apply_with_jac(x.val.val)
13            jac = Pybind11LinearOperator(..., lin)
14            return x.new(pos.position(), jac)
15        return ift.makeField(self.target, self._op.apply(x.val))
16
17 # init op
18 from cpp_for_nifty import SquareOperator as sq_op_cpp
19 sq_op = Pybind11Operator(..., sq_op_cpp)
```

C++ to NIFTy: Summary

C++:

- Apply Operator
- Apply Jacobian
- Apply adjoint Jacobian

pybind11:

- Expose C++ functions to python

NIFTy:

- Interface to NIFTy
- Wrap C++ computations as a nifty operator

C++ to Jax: Summary

C++:

- Apply Operator
- Apply Jacobian
- Apply adjoint Jacobian

pybind11:

- Expose C++ functions to python

NIFTy:

- Interface to Jax
- Wrap C++ computations as a Jax primitive

Jax primitive

- jvp
- vjp
- vmap
- jit
- CPU/ GPU backend

C++ to Jax

```
1 def op(...):  
2  
3 def _op_abstract_eval(...): # for jit  
4  
5 def _op_lowering(...): # for jit  
6  
7 def _op_jvp(...): # for jvp  
8  
9 def _op_transpose(...): # for vjp  
10  
11 def _op_batch(...): # for vmap
```

Tensorflow in Jax

- Jax and Tensorflow both based on XLA
- *jax2tf* module enables interoperability
- Note: At the moment not all jax transforms possible with interfaced tf code
- <https://github.com/google/jax/tree/main/jax/experimental/jax2tf>
- <https://www.tensorflow.org/guide/jax2tf>

Summary

- Interfacing with other programming languages possible
- Binding C++ code for python common practice
- Manual implementation of *jvp* and *vjp* necessary
- DUCC library with many helper functions for handling n-dim arrays in C++:
<https://gitlab.mpcdf.mpg.de/mtr/ducc>
- Interfacing Jax also possible, but a bit experimental
- Two examples for interfacing NIFTy and Jax:
<https://gitlab.mpcdf.mpg.de/jroth/extending-jax-and-nifty>