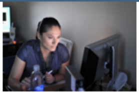
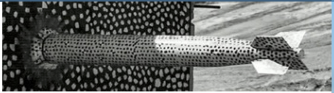
Sandia National Laboratories

PyTrilinos2: automatic (re)generation of a Python interface for Trilinos

CCR Center for Computing Research





Presented by:

K. Liegeois & C. Glusa

Trilinos User Group, October 2023

SAND2023-13641 C

U.S. Department of Energy NNSA

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Motivations for a Python interface for Trilinos

Python is a glue language that allows to:

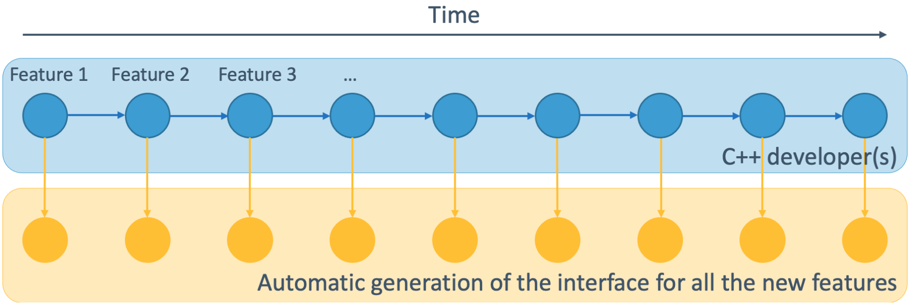
- ease the prototyping of new methods and applications,
- improve the pre and post processing workflow,
- drive complex software through user-friendly front-ends,
- manage inter-operation between libraries and software written in different languages.

Motivations for a Python interface for Trilinos

In particular, a Python interface for Trilinos could allow to:

- drive parametric computations from Python:
  - perform uncertainty quantification analysis,
  - optimize parameter values,
  - compute sensitivity w.r.t parameters using finite differences, ...
- prototype new solvers while relying on existing C++ features:
  - new multigrid cycles for MueLu,
  - new mixed precision linear solvers for Belos,
  - new solvers for Ifpack2, ...
- embed Machine Learning (ML) models written in Python into existing C++ code:
  - use a ML model at an integration point to evaluate as a constitutive model while assembling matrices, ...
- lower the learning curve associated to the usage of Trilinos and potentially increase the user base.

PyTrilinos2: goals of the reboot



Goals:

- Automatic generation of the interface for new packages.
- Automatic update of the interface to reflect C++ changes.
- No latency between new C++ features and their Python interface.

Approach used for PyTrilinos2

PyTrilinos2 relies on the combination of two tools: Pybind11 and Binder.

Pybind11:

- a lightweight header-only library that exposes C++ types in Python and vice versa,
- similar to Boost.Python,
- supports custom smart pointers (such as `Teuchos::RCP`), template arguments, inheritance, ...
- can be easily installed using pip or spack.

Binder:

- tool for the automatic generation of Python bindings using Pybind11 and Clang LibTooling libraries,
- generates the Pybind11 lines in `.cpp` files but does not compile them,
- developed at the Johns Hopkins University by Sergey Lyskov,
- can be installed using spack.

Example of how to add an extra package in the interface

```
#include <PyTrilinos2_Teuchos_ETI.hpp>
#include <PyTrilinos2_Tpetra_ETI.hpp>
#include <PyTrilinos2_MueLu_ETI.hpp> // <---------------------------------- One new header for the new package
```

```
#ifndef PYTRILINOS2_MUELU_ETI
#define PYTRILINOS2_MUELU_ETI
#include <MueLu_CreateTpetraPreconditioner.hpp>

#define BINDER_MUELU_CREATETPETRAPRECONDITIONER_INSTANT(SCALAR,LO,GO,NO) \
  template Teuchos::RCP<MueLu::TpetraOperator<SCALAR, LO, GO, NO> >      \
  CreateTpetraPreconditioner<SCALAR, LO, GO, NO>(                        \
  const Teuchos::RCP<Tpetra::Operator<SCALAR, LO, GO, NO> > &inA,        \
  Teuchos::ParameterList& inParamList);

namespace MueLu {
  BINDER_MUELU_CREATETPETRAPRECONDITIONER_INSTANT(
        Tpetra::Details::DefaultTypes::scalar_type,
        Tpetra::Details::DefaultTypes::local_ordinal_type,
        Tpetra::Details::DefaultTypes::global_ordinal_type,
        Tpetra::KokkosClassic::DefaultNode::DefaultNodeType)
}
#endif // PYTRILINOS2_MUELU_ETI
```

Extra work needed for adding a new package

The binder configuration script needs to be updated to not bind what should not be bound:

```
-class MueLu::FactoryAcceptor
-class MueLu::FactoryFactory
-class MueLu::FactoryManagerBase
-class MueLu::FactoryManager
-class MueLu::FactoryBase
-class MueLu::Hierarchy
#...
```

The PyTrilinos2 CMake files need to be updated to add the new package:

```
#...
list(APPEND BINDER_OPTIONS --bind Teuchos)
list(APPEND BINDER_OPTIONS --bind Tpetra)
list(APPEND BINDER_OPTIONS --bind MueLu) # <---- New line
#...
```

```
#...
SET(LIB_REQUIRED_DEP_PACKAGES
  Teuchos
  Tpetra
  MueLu # <---- New line
  )
#...
```

Example of **automatically** generated bindings for the new package

```
void bind_PyTrilinos2_50(std::function< pybind11::module &(std::string const &namespace_) > &M)
{
{ // MueLu::SingleLevelFactoryBase file:MueLu_SingleLevelFactoryBase.hpp line:64
    pybind11::class_<MueLu::SingleLevelFactoryBase, Teuchos::RCP<MueLu::SingleLevelFactoryBase>, MueLu::Factory>
    cl(M("MueLu"), "SingleLevelFactoryBase", "Base class for factories that use one level (currentLevel).");
}
{ // MueLu::TwoLevelFactoryBase file:MueLu_TwoLevelFactoryBase.hpp line:67
    pybind11::class_<MueLu::TwoLevelFactoryBase, Teuchos::RCP<MueLu::TwoLevelFactoryBase>, MueLu::Factory>
    cl(M("MueLu"), "TwoLevelFactoryBase",
    "Base class for factories that use two levels (fineLevel and coarseLevel).");
}
{ // MueLu::SetFactoryManager file:MueLu_HierarchyUtils_decl.hpp line:65
    pybind11::class_<MueLu::SetFactoryManager, Teuchos::RCP<MueLu::SetFactoryManager>>
    cl(M("MueLu"), "SetFactoryManager", "An exception safe way to call the method 'Level::SetFactoryManager()'");
    cl.def( pybind11::init( [](MueLu::SetFactoryManager const &o){ return new MueLu::SetFactoryManager(o); } ) );
}
// MueLu::CreateTpetraPreconditioner(const class Teuchos::RCP<class ...
  M("MueLu").def("CreateTpetraPreconditioner", (class Teuchos::RCP<class MueLu::TpetraOperator<double, int, long long,
    class Tpetra::KokkosCompat::KokkosDeviceWrapperNode<class Kokkos::OpenMP, class Kokkos::HostSpace> > >
    (*)(const class Teuchos::RCP<class Tpetra::Operator<double, int, long long,
    class Tpetra::KokkosCompat::KokkosDeviceWrapperNode<class Kokkos::OpenMP, class Kokkos::HostSpace> > > &,
    class Teuchos::ParameterList &)) &MueLu::CreateTpetraPreconditioner<double,int,long long,
    Tpetra::KokkosCompat::KokkosDeviceWrapperNode<Kokkos::OpenMP, Kokkos::HostSpace>>,
    "Helper function to create a MueLu or AMGX preconditioner that can be used by Tpetra...",
    pybind11::arg("inA"), pybind11::arg("inParamList"));

}
```

Example of usage of PyTrilinos2

The presented example uses Teuchos, Tpetra, and MueLu.

Why have we started with Tpetra? Tpetra:

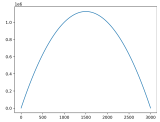
- relies heavily on templates,
- uses MPI communicator,
- is the base package of a stack of solvers,
- relies on Kokkos for shared memory parallelism,
- uses ETI and allows to instantiate multiple instances of one class within one build.

The current work branch of PyTrilinos2 provides binding for `MultiVector`, `Vector`, `Map`, `CrsGraph`, `CrsMatrix`, `Export`, `Import`, ...

Example: 1D Laplacian solved with CG

```python
def main():
    comm = Teuchos.getTeuchosComm(MPI.COMM_WORLD)

    n = 3000
    A = assemble1DLaplacian(n, comm)
    mapT = A.getRowMap()

    x = getTypeName('Vector')(mapT, True)
    b = getTypeName('Vector')(mapT, False)
    b.putScalar(1.)
    pl = Teuchos.ParameterList()
    P = MueLu.CreateTpetraPreconditioner(A, pl)

    CG(A, x, b, P, max_iter=30)
    n0 = 0
    if comm.getRank() == 0:
        n0 = n
    mapT0=type(mapT)(n, n0, 0, comm)

    x0 = getTypeName('Vector')(mapT0, True)
    export = getTypeName('Export')(mapT0, mapT)
    x0.doImport(source=x, exporter=export,\
      CM=Tpetra.CombineMode.REPLACE)

    if comm.getRank() == 0:
        plt.figure()
        plt.plot(x0.getLocalViewHost())
        plt.savefig('x0_view.png')
```

- Solve a 1D Laplacian using CG,
- the CG is written in Python using Tpetra function calls,
- the script can be run with MPI,
- the postprocess is done on the rank 0 after importing the solution vector and getting its local view host (as a NumPy array),
- Kokkos is used under the hood and works with CUDA backend.



PyROL

The same strategy has been apply to ROL in collaboration with **Aurya Javeed**.

Available features:

- The python interface exposes the ROL virtual classes,
- The user can define a child class in Python to define their own vector type.
- The user can define their objective functions in Python.
- The user can then call the optimizer from Python.

PyROL will soon be available and user will be able to install it from source (as PyTrilinos2) or using pip.

```
# Matrix from rol/example/quadratic/example_01.cpp
class matrix(LinearOperator):
    def __init__(self, dim):
        self.dim = dim
        super().__init__()
    def apply(self, Hv, v, tol):
        for i in range(0, self.dim):
            Hv[i] = 2.*v[i]
            if i > 0:
                Hv[i] -= v[i-1]
            if i < self.dim - 1:
                Hv[i] -= v[i+1]

op = matrix(10)
g = vector_type.full(10, 0.)
x = vector_type.full(10, 1.)
zero = vector_type.full(10, 0.)

bnd = Bounds_double_t(zero, x.clone())
obj = QuadraticObjective(op, g)

params = getParametersFromXmlFile("input.xml")
status = StatusTest(params)

problem = Problem_double_t(obj, x)
problem.addBoundConstraint(bnd)
solver = Solver_double_t(problem, params)
cout = openOfstream('test.txt')
solver.solve(cout, status)
closeOfstream(cout)

state = solver.getAlgorithmState()
```

Current stage and future work

Current stage:

- Binder can be installed using spack.
- The approach has been applied on ROL.
- Initial PyTrilinos2 PR is open for review.

Future works:

- The content of the PR will be tested in a nightly build.
- PyTrilinos(1) will be deprecated.
- Create the interface for other packages/features.
- Improve PyTrilinos2 CMake logic to allow optional packages.
- Potentially couple the work with PyKokkos.

Example: Tpetra CG

```
def CG(A, x, b, prec, max_iter=20, tol=1e-8):
    r = type(b)(b, Teuchos.DataAccess.Copy)
    A.apply(x,r,Teuchos.ETransp.NO_TRANS,alpha=-1,beta=1)

    p = type(r)(r, Teuchos.DataAccess.Copy)
    q = type(r)(r, Teuchos.DataAccess.Copy)

    Br = type(r)(r, Teuchos.DataAccess.Copy)
    prec.apply(r, p)
    gamma = sqrt(r.dot(p))

    if gamma < tol:
        return 0
    for j in range(max_iter):
        A.apply(p, q)
        c = q.dot(p)
        alpha = gamma**2 / c
        x.update(alpha, p, 1)
        r.update(-alpha, q, 1)
        prec.apply(r, Br)
        gamma_next = sqrt(Br.dot(r))
        beta = gamma_next**2/gamma**2
        gamma = gamma_next
        if gamma < tol:
            return j+1
        p.update(1, Br, beta)
    return max_iter
```

Example: 1D Laplacian

```
def assemble1DLaplacian(n, comm):
    mapT=getTypeName('Map')(n, 0, comm)
    graph = getTypeName('CrsGraph')(mapT, 3)
    for i in range(mapT.getMinLocalIndex(), mapT.getMaxLocalIndex()+1):
        global_i = mapT.getGlobalElement(i)
        indices = [global_i]
        if global_i > 0:
            indices.append(global_i-1)
        if global_i < mapT.getMaxAllGlobalIndex():
            indices.append(global_i+1)
        graph.insertGlobalIndices(global_i, indices)
    graph.fillComplete()

    A = getTypeName('CrsMatrix')(graph)
    for i in range(mapT.getMinLocalIndex(), mapT.getMaxLocalIndex()+1):
        global_i = mapT.getGlobalElement(i)
        indices = [global_i]
        vals = [2.]
        if global_i > 0:
            indices.append(global_i-1)
            vals.append(-1.)
        if global_i < mapT.getMaxAllGlobalIndex():
            indices.append(global_i+1)
            vals.append(-1.)
        A.replaceGlobalValues(global_i, indices, vals)
    A.fillComplete()
    return A
```