ORNL/TM-2017/171

Existing Fortran interfaces to Trilinos in preparation for exascale ForTrilinos development

Katherine Evans
Mitchell Young
Benjamin Collins
Seth Johnson
Andrey Prokopenko
Mike Heroux (project PI)

**Approved for public release.
Distribution is unlimited.**

March 31, 2017

ORNL/TM-2017/171

Exascale Computing Program

Existing Fortran interfaces to Trilinos in preparation for exascale ForTrilinos development

Katherine Evans (Oak Ridge National Laboratory)
Mitchell Young (Oak Ridge National Laboratory)
Benjamin Collins (Oak Ridge National Laboratory)
Seth Johnson (Oak Ridge National Laboratory)
Andrey Prokopenko (Oak Ridge National Laboratory)
Mike Heroux (project PI) (Sandia National Laboratories)

Date Published: March 2017

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, TN 37831-6283
managed by
UT-Battelle, LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

Existing Fortran interfaces to Trilinos in preparation for exascale ForTrilinos development

An ECP report sponsored by
U.S. Department of Energy, Exascale Computing Program

March 31, 2017

Katherine Evans (Oak Ridge National Laboratory)
Mitchell Young (Oak Ridge National Laboratory)
Benjamin Collins (Oak Ridge National Laboratory)
Seth Johnson (Oak Ridge National Laboratory)
Andrey Prokopenko (Oak Ridge National Laboratory)
Mike Heroux (project PI) (Sandia National Laboratories)

Contents

Executive Summary

This report summarizes the current state of Fortran interfaces to the Trilinos library within several key applications of the Exascale Computing Program (ECP), with the aim of informing developers about strategies to develop ForTrilinos, an exascale-ready, Fortran interface software package within Trilinos. The two software projects assessed within are the DOE Office of Science's Accelerated Climate Model for Energy (ACME) atmosphere component, CAM, and the DOE Office of Nuclear Energy's core-simulator portion of VERA, a nuclear reactor simulation code. Trilinos is an object-oriented, C++ based software project, and spans a collection of algorithms and other enabling technologies such as uncertainty quantification and mesh generation. To date, Trilinos has enabled these codes to achieve large-scale simulation results, however the simulation needs of CAM and VERA-CS will approach exascale over the next five years. A Fortran interface to Trilinos that enables efficient use of programming models and more advanced algorithms is necessary. Where appropriate, the needs of the CAM and VERA-CS software to achieve their simulation goals are called out specifically. With this report, a design document and execution plan for ForTrilinos development can proceed.

1 Introduction

For more than a decade, large scale simulation codes have relied on expertly developed libraries to perform a number of tasks that are common requirements for many applications, but are not easily developed and optimized by domain scientists, including solutions to partial differential equations, eigenvalue algorithms, parallel programming, I/O support, and more. Trilinos [6] is one example of a suite of algorithm and other technologies that allow codes to point to optimized and generalized software for scalable simulation. Being C++ based, using Trilinos within a Fortran code motivated the development of ForTrilinos [8], a Trilinos package to handle access to Trilinos, including common data structures with iso-c-binding features within Fortran 2003. In order to develop an advanced ForTrilinos interface that handles a diversity of code applications and structures, a new ForTrilinos project that is a part of the Exascale Computing Project (ECP), has been initiated. It targets 2 codes that are important for the implementation of the new software capability. This will provide robust testing and evaluation, and inform the progress and success of ForTrilinos.

The first target code is the Accelerated Climate Model for Energy (ACME) atmosphere model (CAM). The DOE ACME project [2] has outlined ambitious simulation goals as part of its mission to efficiently utilize DOE leadership computing resources now and in the future. Specifically, DOE BER and ASCR have invested in a global coupled Earth system model that will run problems of relevance to DOE, targeting exascale machines. There is an ACME ECP applications project, ACME-MMF, which is targeting superparameterization for an exascale application. Even though they are targeting another part of the ACME codebase, enabling ForTrilinos in the dynamics, where an existing ForTrilinos interface exists, could be extended to other parts of ACME where related work is occurring, leveraging work across both projects.

We also target a code within the stable of simulation tools within the Consortium for Advanced Simulation of LWRs (CASL), the Virtual Environment for Reactor Applications, VERA. CASL is a DOE Modeling and Simulation Hub, which uses leading-edge modeling and simulation (M&S) capability to improve the performance of currently operating light water reactors. CASL's vision is to enable safer and more efficient production of commercial nuclear power through comprehensive, science-based predictive M&S technology deployed and applied broadly by the U.S. nuclear energy industry. As with ACME, there is also ECP applications project, ExaSMR, that is focused on other aspects of the same problem of nuclear reactor simulation. Having ECP efforts devoted to multiple aspects of exascale diversifies efforts and allows existing codes to achieve their simulation goals within the larger DOE mission.

This report is designed to provide a resource of the details of the current interfaces of the two Fortran-based target codes so that ForTrilinos developers can develop a design plan. This is not meant to include every detail about the code and their interactions of Fortran and C++, but provide enough information to motivate the necessary work to be done, and links to the details to flesh out the work plan. Section 2 and 3 explain the interfaces for CAM and VERA-CS respectively, and section 4 provides a quick summary.

2 Global Atmospheric Modeling

2.1 Overview Description and Impact

As part of the ForTrilinos project, we will enable the ACME atmosphere model (CAM) implicit solver of the fluid flow to run with demonstrated and sufficient efficiency and scalability on Summit. The solver currently utilizes a testbed ForTrilinos interface to Trilinos packages for its solution [3, 7], and the work to extend it for production use within the full hydrostatic equations as part of a BER SciDAC is ongoing. Efforts to utilize the hybrid CPU-GPU Titan supercomputer at OLCF has shown significant speed-up of the residual calculation within the solver using CUDA Fortran [1], however the interface between the Fortran based residual and the C++ based Trilinos solvers prevents the data from remaining on the GPU between iterations, which more than removes the benefit of speed-up gained from the computation within the residual. With a ForTrilinos capability that enables data structures to be maintained across C++ based Trilinos and Fortran based applications, the overall performance gain within the solver, using the Fortran-based domain residual and preconditioner code, is straightforward and substantial. Furthermore, if these common data structures can allow for generality and portability across architectures, then ACME can utilize those systems effectively.

2.2 System Requirements

Our software within ACME-CAM uses several compilers on OLCF Titan, although the GPU implementation within the solver requires that we use PGI at this time. PGI is generally 20% slower than GNU, even with some effort to optimize the build. CAM uses Fortran 2003 iso-C-bindings to interface the data structures with the C++ interface to Trilinos. Right we use NOX, Belos and Teuchos, and we will use Kokkos, extending it as needed, perhaps as part of ForTrilinos to create the generic data structures to transfer the data across the language barrier. The GPU implementation used CUDA Fortran, but has now transitioned to OpenACC [10] to achieve greater portability going forward. It is an unsolved issue as to whether CUDA or OpenACC is the best option for mixed language code, and how that selection will be made. The build of CAM with Trilinos and using GPU is currently very complicated, fragile, and error-prone; efforts to harden the process for production will be required. These efforts will require interactions with the software developers of CAM, Trilinos and the compiler and computing facility staff.

The performance of CAM using an implicit solver of the fluid dynamical core using Trilinos has been analyzed, and optimization to make it more efficient than the traditional explicit solver is ongoing. An analysis of the method in 2D (so no preconditioner) to motive the efforts in the production hydrostatic code shows that (1) refined grid configurations produce relatively more efficient solutions compared to explicit due to the time step size restriction, (2) using a higher order configuration of the spectral element discretization of CAM increases the intensity of computation within a node for a given parallel decomposition, and may provide more efficient computation on hybrid CPU-GPU systems, and (3), there are many solver parameters that significantly affect performance (e.g. choice of orthogonalization), and access to these by the domain scientists at run time, without rebuilding Trilinos, will improve production usage. CUDA is faster than OpenACC, but OpenACC continues to close the gap as we learn how to use it and as it matures. We have added C-based GPTL performance timers to both the Fortran application and Trilinos code to track performance. We will require support from the compilers and software vendors on this machine going forward.

2.3 Current interface structure

The current interface to enable Fortran/C++ interaction within CAM and a related climate application the Community Ice Sheet Model (CISM1.0) follows the developments in ForTrilinos 1.0, and the details are provided elsewhere [3, 4]. CAM supports the use of the C++ based Trilinos library within this interface. From Trilinos, CAM uses the NOX nonlinear solver package to perform the nonlinear solve for a given time step size, which is passed to it from the Fortran code.

To maintain backwards capability and allow users of the code to build and run without Trilinos, CAM within ACME supports multiple build options for new solvers and machine implementations using the cmake target feature, thanks to Matt Norman (personal communication, and ACME documentation). The top level directory, `src/`, has files specific for that solver and `src/share` will host files for the default scheme. All solvers/cmake targets must overwrite the same modules. If a new module is needed, a new blank module is created for all versions, and the default is blank. This replaces the `#ifdef` preprocessor structure used in the references above.

The code to create the interface is initialized and finalized in outer routines by creating the necessary derived types and pointers and initializing and finalizing everything for NOX as follows:

```
    use, intrinsic :: iso_c_binding
    integer :: lenx
    real (c_double) ,allocatable ,dimension(:) :: xstate

! state_object is a derived data type passed thru noxinit as a pointer
    type(derived_type) ,target         :: state_object
    type(derived_type) ,pointer        :: fptr=>NULL()
    type(c_ptr)                        :: c_ptr_to_object
! fortran subroutine to wrap up all variables needed by the IOC into a derived type
    call initialize(state_object, lenx, elem, pmean,edge1,edge2, edge3, &
      hybrid, deriv, dt, tl, nets, nete)
    fptr => state_object
    c_ptr_to_object =  c_loc(fptr)
    call noxinit(size(xstate), xstate, 1, c_ptr_to_object, c_ptr_to_pre)

    (main code)

    call noxfinish()
    deallocate(xstate)
```

The Fortran code (located in `src/share/prim_driver_mod.F90`) accesses Trilinos through several C++ modules within the code base (located in `utils/trilinos`). Then, subroutines that are used within the C++ code to connect to Trilinos are also called in Fortran via an interface:

```
  interface
   subroutine noxsolve(vectorSize,vector,v_container,p_container,j_container,ierr) &
     bind(C,name='noxsolve')
    use ,intrinsic :: iso_c_binding
      integer(c_int)                 :: vectorSize
      real(c_double)  ,dimension(*) :: vector
      type(c_ptr)                    :: v_container
      type(c_ptr)                    :: p_container  !precon ptr
      type(c_ptr)                    :: j_container  !analytic jacobian ptr
      integer(c_int)                 :: ierr         !error flag
    end subroutine noxsolve
  end interface
```

The interface exists for 3 routines, `noxinit`, `noxfinalize`, and `noxsolve`. Then within each time step (in CAM this is in the `prim_advance_mod.F90` module), a flat state vector, `xstate` of length `lenx` is populated from a multidimensional state vector in CAM, and then `noxsolve` is called within the main CAM time stepping loop to solve for the next time level:

```
   call noxsolve(size(xstate), xstate, c_ptr_to_object, c_ptr_to_pre, c_ptr_to_jac, ierr)
```

After its complete, the `xstate(lenx)` repopulates the multidimensional vector with the updated values at the next time level. The `noxsolve` routine in C++ coordinates the work of solving the nonlinear equations via callbacks to the Fortran to calculate a residual for each linear update and to assess convergence. The residual function is calculated in Fortran, with a call from C++ (in CAM this is in the `/utils/trilinos/trilinosNoxSolver.cpp` module):

```
void calc_f(double *, double *, int, void *);     // residual
void (*residualFunction)(double *, double *, int, void *) = calc_f;
```

which exists in the Fortran as:

```
subroutine residual(xstate, fx, nelemd, c_ptr_to_object) &
  bind(C,name='calc_f').
```

All the NOX infrastructure is completed within a LOCA interface, so that we can use parameter continuation for convergence assistance. The Loca, NOX, and Stratimikos settings are given by the user at runtime via a `trilinosoptions.xml` file, which is read by the C++ interface files.

The C++ interface also handles the linear solver details using Stratimikos, which is targeted for improvement within ForTrilinos to allow the Fortran to handle instead. Corresponding callbacks from the C++ to the Fortran are also performed to operate on the blocks of the preconditioner within GMRES. So the preconditioner relies on existing code in the Fortran to evaluate the blocks. The details of the preconditioner as currently implemented within the shallow water version of CAM, our initial target, are provided in Lott et al. [7].

2.4 Cross-team Collaboration/Integration

We already have a team is in place to work on ForTrilinos in CAM, thanks to our ongoing efforts to develop and optimize the Trilinos based implicit solver in CAM and our connections to Trilinos developers that worked with us to develop the initial ForTrilinos interface. We attended an OLCF Hackathon in the fall of 2015 to learn how to include OpenACC in the Fortran code. The team includes computational climate scientists that develop and analyze algorithms within ACME, a domain scientist to perform model evaluation and assessment that will determine how well ACME captures the targeting climate features resolved at fine scales. New to our team are the addition of Trilinos, Fortran, and C++ experts to help us in using Kokkos within the Trilinos solver and to make the connections to the Fortran data structures using Kokkos. We already have a suite of build options of our code using Trilinos and can apply the executables to domain approved test cases to demonstrate capability and performance benchmarks.

2.5 Related Research

The largest barrier for ACME success is to maintain model throughput while adding new model features and resolution. BER SciDAC support to develop the solver itself is ongoing through FY17, whereby a full implicit solver within ACME will be available. But as current work demonstrates [5], use of the GPU through the Trilinos solver awaits an common data structure of the state vector, residual, and preconditioner matrix to keep the data on the GPU and prevent the transfer bottleneck. Although this would demonstrate success for the fluid flow solver in CAM, the tracer advection and vertical remap would also benefit from Trilinos solvers, as would other parts of components within ACME, that use Trilinos (e.g. ice sheets [4], or could use Trilinos, for example the MPAS ocean model (Trilinos was implemented within a test version of the previous ocean ”POP” model [9]).

2.6 Other Considerations/Issues

One unknown and therefore source of risk is the performance profile of ACME as it is developed during the course of the project. For example, ACME version 1 due to be released in FY2017 will contain about double the number of vertical levels as compared to version 0, so the intensity of computation should increase. There are many other features with measurable impacts of which we should be aware. The risks also involve the degree to which the data can remain on the GPU throughout the cycle of a complex nonlinear preconditioned solver with multiple callbacks through the different code pieces and the degree of minimized use of MPI.

We achieved a lot of progress by attending the OLCF Hackathon last year, and so we would recommend this team attend one together this year, and include some Kokkos developers as well. We require a much better, more stable build process and so interactions with the compiler teams and LCF are encouraged. Training in C++ and Trilinos for the applications teams here at ORNL is highly recommended.

3 Deterministic Neutron Transport Simulation

3.1 Overview Description and Impact

CASL's VERA software simulates nuclear reactor physical phenomena using coupled multi-physics models. VERA's current physics capabilities include deterministic and stochastic neutron transport, thermal-hydraulics, fuel performance, and coolant chemistry. The nuclear reactor core simulator component of CASL, VERA-CS, is comprised of a deterministic neutron transport code, MPACT, and a subchannel thermal-hydraulic code, COBRA-TF. MPACT is a new code written entirely in Fortran 2003. COBRA-TF is also written entirely in Fortran and has been around for decades; within CASL, COBRA-TF is being modernized from Fortran 77 to Fortran 2003. Both codes leverage parallel linear solvers in order to obtain the coupled multiphysics state of a nuclear reactor through the lifetime of the fuel. Currently MPACT and COBRA-TF both use PETSc linear solvers because of the availability of a supported Fortran interface. There are multiple reasons why Trilinos is a more attractive option. Namely, the availability of the ML preconditioner, access to a native Eigenvalue solver class, and the ability to extend these solvers for multithreaded and many-core platforms. A ForTrilinos interface would expose functionality available in Trilinos that can significantly impact the performance of VERA-CS.

3.2 System Requirements

The VERA-CS software stack requires a Fortran 2003 compliant compiler and a version 2.0 compliant distribution of MPI. Regular testing is performed on both GNU and Intel compilers (due to current limitations in Fortran 2003 support, the PGI compiler cannot be used). Ports of MPACT to Cray, Pathscale and IBM XL compilers have also been attempted, but these compilers still have issues in their support of Fortran 2003. Because VERA-CS is designed for use by the nuclear industry, which has limited computational resources, the target platform for a single VERA-CS calculation is a moderate cluster, from hundreds to thousands of cores, solving problems in less than 24 hours (50,000 CPU-hours). However, the highly regulated nuclear industry requires uncertainty estimates on all simulations, which necessitates hundreds to thousands of calculations utilizing the tools available within Dakota, a UQ tool in the Trilinos software package. For example, CASL will be modeling a single cycle of the Watts Bar Nuclear Power Plant to predict, with uncertainty bounds, the performance of the reactor during an operational transient that occurred in 2005. This single simulation, planned for late FY2015, will require over 10 million CPU-hours on Titan to obtain a rough estimate of the uncertainty. For CASL to provide a rigorous uncertainty estimate, the number of VERA-CS calculations will be 1-2 orders of magnitude higher.

There is currently an exploratory effort within CASL to enable the transport sweep algorithm within MPACT to effectively utilize the GPU technology. However, currently the computational burden in the target applications is limited by the PETSc eigenvalue, linear, and preconditioner evaluations of the large, sparse matrices of both COBRA-TF and MPACT. By developing modern Fortran interfaces to the solvers within Trilinos and creating a link to the Kokkos programming model, CASL will be able to effectively leverage the full power of Summit and dramatically improve the computational efficiency of VERA-CS through efficient use of the GPUs. This will provide a dual benefit: demonstrating the use of Leadership Computing Facility capabilities to the nuclear industry for providing efficient prediction of reactor performance with uncertainty bounds and greatly improving the penetration of CASL software within the nuclear industry through efficient use of modern clusters that leverage GPU technology.

3.3 Technology Maturity

The main focus of the VERA-CS team over the past few years has been to develop a capability that can accurately simulate LWRs for multiple fuel cycles. VERA-CS has been verified with extensive testing and validated through comparison with experimental facilities and 18 years of operation of the Watts Bar Nuclear Power Plant. The validation includes comparisons with startup testing measurements every cycle (18 months) and neutron flux detector measurements during operation. VERA-CS is currently deployed to multiple industry partners in the U.S. and South Korea for evaluation and additional validation.

There are multiple current CASL milestones to improve the computational performance of every component of VERA-CS. Current improvements this year have resulted in an approximately 2.5x speedup in VERA-CS runtime on traditional CPUs. Several more improvements are planned with the expectation of an overall 8x speedup by the end of the fiscal year. Based on some scoping studies, the use of Anasazi with a ML preconditioner could result in another 1.5x speedup over the existing improvements.

Currently a few small scale studies have investigated porting the MPACT solver to GPUs but a significant effort has not been the focus of current development in VERA-CS. However, it is clear that this leap will be required to enable the scale of simulation needed to satisfy CASL's uncertainty quantification plans.

3.4 Current interface structure

A number of Trilinos packages are already being used in VERA-CS, including the Anasazi package along with the ML and Ifpack preconditioner packages. These packages are supported by the Epetra and Teuchos packages, which provide the necessary matrix, vector, and parameter list classes. All of these packages are wrapped by hand using an ad hoc approach designed to make the interface between the native VERA-CS Fortran code and the Trilinos libraries as narrow as possible. Allocation of Trilinos objects is handled on the C++ side, with objects being stored in global `std::map<int, T>` stores. The entries of the maps are structs containing all of the objects needed to use the desired solver. New objects are allocated by calling Fortran-wrapped `extern "C"` functions that insert a new object into the map, and return the corresponding index. Fortran code can then use this index to interact with the object, all of which must also be exposed through `extern "C"` routines.

For example, the Anasazi solvers are stored in a `std::map<int,AnasaziCnt>` container, where `AnasaziCnt` is a struct composing all of the components needed to perform a generalized eigenvalue solve:

```
struct AnasaziCnt{
    //We're solving   LHS*x = k*RHS*x
    Teuchos::RCP<Epetra_CrsMatrix> LHS;
    Teuchos::RCP<Epetra_CrsMatrix> RHS;
    Teuchos::RCP<Epetra_Operator>  pc;
    bool haspc=false;
    double keff;
    int niters;
    int pc_id;
    Teuchos::RCP<Epetra_Vector> x;
    Teuchos::ParameterList anasazi_db;
};

std::map<int, AnasaziCnt> anasazi_map;
```

Functionality is then exposed to with `extern "C"` interfaces, which allow for Fortran code to drive the Trilinos solvers.

```
//---------------------------------------------------------------------------
// Anasazi C wrappers
//---------------------------------------------------------------------------
extern "C" {
void Anasazi_Init(int &id, CTeuchos_ParameterList_ID &plist) {
    auto plistDB = CTeuchos::getNonconstParameterListDB();
    Teuchos::Ptr<Teuchos::ParameterList> params =
        plistDB->getNonconstObjPtr(plist.id);
    //setup parameterlist with defaults
    params.set("Which", std::string("LM"));
    params.get("Convergence␣Tolerance",1e-7);
    params.get("Maximum␣Subspace␣Dimension",25);
    params.get("Restart␣Dimension",5);
    params.get("Maximum␣Restarts",20);
    params.get("Initial␣Guess",std::string("User"));
    params.get("Verbosity",Anasazi::Errors + Anasazi::Warnings);

    // Generate a new index one past the largest.
    id = anasazi_map.rbegin()->first+1;
    anasazi_map[id] = AnasaziCnt();
    anasazi_map[id].anasazi_db = params;
}

void Anasazi_Destroy(const int id) {
    anasazi_map.erase(id);
}

void Anasazi_SetMat(const int id, const int idLHS, const int idRHS) {
    anasazi_map[id].LHS = epetra_matrix_map[idLHS];
    anasazi_map[id].RHS = epetra_matrix_map[idRHS];
}

void Anasazi_SetPC(const int id, const int idpc) {
    anasazi_map[id].pc = pc_map[idpc];
    anasazi_map[id].haspc = true;
    anasazi_map[id].pc_id = idpc;
}

void Anasazi_SetX(const int id, const int idX) {
    anasazi_map[id].x = epetra_vec_map[idX];
}
```

```
void Anasazi_SetConvCrit(const int id, const double tol,
                                    const int maxit) {
    anasazi_map[id].anasazi_db.set("Convergence␣Tolerance", tol);
    anasazi_map[id].anasazi_db.set("Maximum␣Restarts", maxit);
}

void Anasazi_Solve(int id) {
    // get a reference to the indexed Anasazi container
    auto &anasazi = anasazi_map[id];

    Teuchos::RCP<Anasazi::BasicEigenproblem<double,Epetra_MultiVector,
                                            Epetra_Operator>>
        problem(new Anasazi::BasicEigenproblem<double,Epetra_MultiVector,
                                               Epetra_Operator>());
    problem->setA(anasazi.LHS);
    problem->setM(anasazi.RHS);
    if(anasazi.haspc) problem->setPrec(anasazi.pc);
    problem->setInitVec(anasazi.x);
    problem->setNEV(1);
    bool problem_set = problem->setProblem();
    assert(problem_set);

    Anasazi::GeneralizedDavidsonSolMgr<double,Epetra_MultiVector,
                                       Epetra_Operator>
        solver(problem, anasazi.anasazi_db);

    Anasazi::ReturnType returnval = solver.solve();

    if(returnval==Anasazi::Converged){
        anasazi.niters=solver.getNumIters();

        // Extract solution
        Anasazi::Eigensolution<double,Epetra_MultiVector> solution =
            solver.getProblem().getSolution();
        Anasazi::Value<double> eval = (solution.Evals)[0];
        anasazi.keff = eval.realpart;
        double val[0];
        solution.Evecs->MeanValue(val);
        anasazi.x->Update(1.0/val[0],*(solution.Evecs),0.0);
        return 0;
    } else {
        //If Anasazi doesn't return, approximate k as x^T Fx/x^M Fx
        anasazi.niters=-1;
        double rhs=1.0;
        double lhs=1.0;
        Epetra_Vector tmp=Epetra_Vector(*(anasazi.x));
        anasazi.RHS->Multiply(false,*(anasazi.x),tmp);
        tmp.Dot(*(anasazi.x),&rhs);
        anasazi.LHS->Multiply(false,*(anasazi.x),tmp);
        tmp.Dot(*(anasazi.x),&lhs);
        //LHS*phi = k * RHS * phi
        anasazi.keff=lhs/rhs;
        return 1;
    }
}

void Anasazi_GetEigenvalue(const int id, double &k) {
    k = anasazi_map[id].keff;
}

void Anasazi_GetResid(const int id, double &resid) {
    Teuchos::RCP<Epetra_Vector> ltmp(new Epetra_Vector(*anasazi_map[id].x));
    Teuchos::RCP<Epetra_Vector> rtmp(new Epetra_Vector(*anasazi_map[id].x));
    anasazi_map[id].LHS->Multiply(false,*(anasazi_map[id].x),*ltmp);
    anasazi_map[id].RHS->Multiply(false,*(anasazi_map[id].x),*rtmp);
    double denom[1];
    (anasazi_map[id].x)->Norm2(denom);
    rtmp->Update(-1.,*(ltmp),anasazi_map[id].keff);
    double resids[1];
    rtmp->Norm2(resids);
    resid=resids[0]/denom[0];
}

void Anasazi_GetIterationCount(const int id, int &niter) {
    anasazi_map->getIterations_data(id, niter);
}
}
```

Similar interfaces are provided for the Epetra vector and matrix types, the Teuchos parameter list, the Belos linear solvers, Ifpack and ML preconditioners, etcetera. On the Fortran side, interfaces to the `extern "C"` functions are specified to allow them to be called as foreign functions:

```
!-------------------------------------------------------------------------------
! Anasazi Fortran Interfaces
!-------------------------------------------------------------------------------
SUBROUTINE Anasazi_Init_Params(id, plID) bind(C,NAME="Anasazi_Init")
  IMPORT :: C_INT
  IMPORT :: ForTeuchos_ParameterList_ID
  INTEGER(C_INT),INTENT(INOUT)    :: id
  TYPE(ForTeuchos_ParameterList_ID), INTENT(IN) :: plID
ENDSUBROUTINE

SUBROUTINE Anasazi_Destroy(id) bind(C,NAME="Anasazi_Destroy")
  IMPORT :: C_INT
  INTEGER(C_INT),INTENT(IN),VALUE  :: id
ENDSUBROUTINE

SUBROUTINE Anasazi_SetMat(id,idLHS,idRHS) bind(C,NAME="Anasazi_SetMat")
  IMPORT :: C_INT
  INTEGER(C_INT),INTENT(IN),VALUE  :: id
  INTEGER(C_INT),INTENT(IN),VALUE  :: idLHS
  INTEGER(C_INT),INTENT(IN),VALUE  :: idRHS
ENDSUBROUTINE

SUBROUTINE Anasazi_SetPC(id,idPC) bind(C,NAME="Anasazi_SetPC")
  IMPORT :: C_INT
  INTEGER(C_INT),INTENT(IN),VALUE  :: id
  INTEGER(C_INT),INTENT(IN),VALUE  :: idPC
ENDSUBROUTINE

SUBROUTINE Anasazi_SetX(id,idX) bind(C,NAME="Anasazi_SetX")
  IMPORT :: C_INT
  INTEGER(C_INT),INTENT(IN),VALUE  :: id
  INTEGER(C_INT),INTENT(IN),VALUE  :: idX
ENDSUBROUTINE

SUBROUTINE Anasazi_SetConvCrit(id,tol,maxit) &
```

```
    bind(C,NAME="Anasazi_SetConvCrit")
  IMPORT :: C_INT,C_DOUBLE
  INTEGER(C_INT),INTENT(IN),VALUE  :: id
  REAL(C_DOUBLE),INTENT(IN),VALUE  :: tol
  INTEGER(C_INT),INTENT(IN),VALUE  :: maxit
ENDSUBROUTINE

SUBROUTINE Anasazi_Solve(id) bind(C,NAME="Anasazi_Solve")
  IMPORT :: C_INT
  INTEGER(C_INT),INTENT(IN),VALUE  :: id
ENDSUBROUTINE

SUBROUTINE Anasazi_GetEigenvalue(id,k) bind(C,NAME="Anasazi_GetEigenvalue")
  IMPORT :: C_INT,C_DOUBLE
  INTEGER(C_INT),INTENT(IN),VALUE  :: id
  REAL(C_DOUBLE),INTENT(OUT)       :: k
ENDSUBROUTINE

SUBROUTINE Anasazi_GetResid(id,resid) bind(C,NAME="Anasazi_GetResid")
  IMPORT :: C_INT,C_DOUBLE
  INTEGER(C_INT),INTENT(IN),VALUE  :: id
  REAL(C_DOUBLE),INTENT(OUT)       :: resid
ENDSUBROUTINE

SUBROUTINE Anasazi_GetIterationCount(id,niter) &
    bind(C,NAME="Anasazi_GetIterationCount")
  IMPORT :: C_INT
  INTEGER(C_INT),INTENT(IN),VALUE  :: id
  INTEGER(C_INT),INTENT(OUT)       :: niter
ENDSUBROUTINE
```

To improve ergonomics in the rest of the code, these simple interfaces are wrapped in a Fortran derived type, which drives them with `init()` and `solve()` type-bound procedures. Other Trilinos components used by the Anasazi solver (Epetra matrix and vector, ML preconditioner) are wrapped in a similar manner and used by this derived type:

```
TYPE :: EigenvalueSolverType_Anasazi
!>
!> size of eigenvalue system
INTEGER(SIK) :: n=-1
!> MPI communicator
TYPE(MPI_Comm) :: MPIComm
!> eigenvalue of the system
REAL(SRK) :: k
!> Setup PC as part of %solve()?
LOGICAL(SBK) :: setupPC=.TRUE.
!> Index into anasazi_map
INTEGER(SIK) :: eig
!> Index into pc_map
INTEGER(SIK) :: pc
!> Pointer to the MatrixType A (wraps index into epetra_mat_map)
CLASS(MatrixType),POINTER :: A => NULL()
!> Pointer to the MatrixType B (wraps index into epetra_mat_map)
CLASS(MatrixType),POINTER :: B => NULL()
!> Pointer to solution vector, x  (wraps index into epetra_vec_map)
CLASS(VectorType),POINTER :: X

!> store vector for scaling fission source (wraps index to epetra_vec_map)
TYPE(TrilinosVectorType) :: x_scale
```

```
!
!List of Type Bound Procedures
CONTAINS
  PROCEDURE,PASS :: init
  PROCEDURE,PASS :: clear
  PROCEDURE,PASS :: solve
ENDTYPE EigenvalueSolverType_Anasazi

!...

SUBROUTINE init(solver,MPIComm,Params)
  CHARACTER(LEN=*),PARAMETER :: myName='init_EigenvalueSolverType_Anasazi'
  CLASS(EigenvalueSolverType_Anasazi),INTENT(INOUT) :: solver
  TYPE(MPI_Comm),INTENT(IN),TARGET :: MPIComm
  TYPE(ParamType),INTENT(IN) :: Params
  TYPE(ParamType) :: tmpPL
  INTEGER(SIK) :: n,nlocal,ierr
  CLASS(ParamType),POINTER :: anasaziParams, pcParams
  TYPE(ForTeuchos_ParameterList_ID) :: plID

  n=0
  nlocal=0

  solver%MPIComm=MPIComm

  !Pull Data from Parameter List
  CALL Params%get('EigenvalueSolverType->n',n)
  CALL Params%get('EigenvalueSolverType->nlocal',nlocal)

  solver%n=n

  !Convert the internal parameter list format to a Trilinos parameter list
  CALL Params%get('EigenvalueSolverType->anasazi_options', anasaziParams)
  plID = Teuchos_ParameterList_Create(ierr)
  CALL anasaziParams%toTeuchosPlist(plID)
  !Initialize the Anasazi solver
  CALL Anasazi_Init(solver%eig, plID)
  CALL Teuchos_ParameterList_Release(plID, ierr)

  IF(solvertype/=GD) THEN
    RaiseError('Incorrect␣input␣to␣'//modName//'::'//myName// &
        '␣-␣Only␣Generalized␣Davidson␣works␣with␣Anasazi.')
  ENDIF

  !Initialize preconditioner. Currently hard-coded to 2, which maps to ML
  CALL Preconditioner_Init(solver%pc,2)

  ALLOCATE(TrilinosVectorType :: solver%X)
  CALL tmpPL%add('VectorType->n',n)
  CALL tmpPL%add('VectorType->MPI_Comm_ID',solver%MPIComm)
  CALL tmpPL%add('VectorType->nlocal',nlocal)
  CALL solver%X%init(tmpPL)
  CALL solver%X_scale%init(tmpPL)
  SELECTTYPE(x=>solver%X); TYPE IS(TrilinosVectorType)
    CALL Anasazi_SetX(solver%eig,x%b)
  ENDSELECT

  solver%TPLType=Anasazi
ENDSUBROUTINE init
```

```
SUBROUTINE solve(solver)
  CLASS(EigenvalueSolverType_Anasazi),INTENT(INOUT) :: solver
  REAL(SRK) :: factor
  REAL(SRK) :: tmp(2)

  ! The matrices solver%A and solver%B are initialized and associated elsewhere
  SELECTTYPE(A=>solver%A); TYPE IS(TrilinosMatrixType)
    SELECTTYPE(B=>solver%B); TYPE IS(TrilinosMatrixType)
      IF (.NOT.(A%isAssembled)) CALL A%assemble()
      IF (.NOT.(B%isAssembled)) CALL B%assemble()
      IF(solver%setupPC) THEN
        CALL Preconditioner_Setup(solver%pc,B%A)
        solver%setupPC=.FALSE.
      ENDIF
      CALL Anasazi_SetMat(solver%eig,A%A,B%A)
    ENDSELECT
  ENDSELECT

  CALL Anasazi_SetPC(solver%eig,solver%pc)
  CALL Anasazi_Solve(solver%eig)
  CALL Anasazi_GetEigenvalue(solver%eig,solver%k)
ENDSUBROUTINE solve

! clear subroutine omitted for brevity
```

Some simplifications have been made in the above code, to make them more presentable in this context. The interfaces in their entirety can be found as part of the Futility library, which can be found at https://github.com/CASL/Futility/. The Trilinos wrappers specifically are at https://github.com/CASL/Futility/tree/master/src/trilinos_interfaces.

This approach, while functional, has a number of drawbacks. First, all of these wrappers were written by hand, which is time-consuming and error-prone; future changes would be similarly onerous. Second, in the interest of keeping the interface with the client Fortran code small, much of the logic associated with the Trilinos objects is implemented in C++, rather than in Fortran. This not only makes it difficult for Fortran developers to understand what is happening under the hood, but is also application-specific. For instance, the code in the `Anasazi_Solve()` function is specific to the type of eigenvalue solution sought by VERA, manually selecting the `GeneralizedDavidsonSolMgr` to solve the eigenproblem, and extracting results from the converged solution specific to the needs of the client code. Under a different context, the bulk of the Anasazi wrappers would need to be re-written (or more realistically duplicated and modified) to accommodate a different use-case. Such duplication and modification presents a significant challenge to code maintainability. More general Fortran bindings, such as those proposed by the ForTrilinos project would allow for much of the problem-dependent logic to be implemented directly in Fortran, eliminating the need for any C++ code.

3.5 Cross-team Collaboration/Integration

The CASL program includes 10 core partners that includes a diverse collection of researchers, computer scientists, and application engineers. VERA-CS is one portion of VERA, which is developed by a collection of researchers at many institutions, but has major portions led by several members of the Trilinos development team at Sandia National Laboratories, including Roger Pawlowski (Coupling Methods Development) and Ross Bartlett (Software Infrastructure). These existing partnerships will ensure effective collaboration and communication within the project.

3.6 Related Research

VERA-CS is under active development inside CASL and is being deployed to the nuclear industry and other laboratory and university partners. There is considerable interest inside and outside of CASL to improve the computational efficiency of VERA-CS. The availability of ForTrilinos will help VERA-CS meet the performance goals desired by its user-base.

4 Conclusions

The two target codes, CAM and VERA-CS, both use C++ interfaces to access the C++ based Trilinos solvers. However that is limiting their ability to access GPU as well as provide a more robust and stable code base. There are several issues that are already apparent within these existing interfaces that will need to be addressed with ForTrilinos. For example, while codes on Titan require the use of PGI to utilize GPU, in particular CAM uses it when using the OpenACC directives within the Fortran residual solver that is called within Trilinos, the VERA code cannot use PGI because some of the Fortran 2003 standards used by the code have not been implemented into PGI. So this limitation needs to be addressed. CAM uses Epetra rather than Tpetra for its classes, and Kokkos and many other more recently developed Trilinos packages use Tpetra, so CAM will need to be udpated. This may be a trivial change within other already occurring changes, but it needs to be considered within the design document. The Tpetra setup in VERA may lead the way for this.

In addition to several issues that have been uncovered through this assessment of the current interfaces, several opportunities to leverage work on each code to benefit the other show potential. For example, the requirements for those who use VERA and CAM to create uncertainty bounds of their simulations strongly motivates the use of Dakota, a Trilinos UQ package, within ForTrilinos. Both codes also highlight the use of GPUs within Trilinos so the Kokkos efforts within ForTrilinos are a high priority. Using these existing interfaces and their attributes and complexities as a guide, ForTrilinos developers will develop a design document and share the document with other interested parties to inform and solicit ideas. Then, ForTrilinos developers will execute the plan.

Acknowledgments

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. This manuscript has been authored by UT-Battelle, LLC and used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, both of which are supported by the Office of Science of the U.S. Department of Energy under Contract No.DE-AC05-00OR22725. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

References

[1] R. K. Archibald, K. J. Evans, and A. Salinger. Accelerating time integration for climate modeling using GPUs. *Procedia Computer Science*, 51:2046–2055, 2015. doi:10.1016/j.procs.2015.05.470.

[2] Bader, D.B. et al. ACME Documentation. http://climatemodeling.science.energy.gov/acme/information-for-collaborators.

[3] K. Evans, D. Rouson, A. Salinger, M. Taylor, J. B. W. III, and W. Weijer. A scalable and adaptable solution framework for components of the Community Climate System Model. *Lecture Notes in Comp. Sci.*, 5545:332–341, 2009. doi:10.1007/978-3-642-01973-9.

[4] K. J. Evans, A. G. Salinger, P. Worley, S. F. Price, W. H. Lipscomb, J. A. Nichols, J. B. W. III, M. Perego, M. Vertenstein, J. Edwards, and J.-F. Lemieux. A modern solver interface to manage solution algorithms in the community earth system model. *Internat. J. High Perf. Comput. Appl.*, 26:54–62, 2012. doi:10.1177/1094342011435159.

[5] K. J. Evans, R. K. Archibald, D. Gardner, M. R. Norman, M. A. Taylor, C. Woodward, and P. Worley. Performance analysis of fully-explicit and fully-implicit solvers within a spectral-element shallow-water atmosphere model. *Internat. J. High Perf. Comput. Appl.*, Submitted., 2017.

[6] M. A. Heroux, R. A. Bartlett, V. E. Howe, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, and A. Williams. An overview of the Trilinos project. *ACM Trans. Math. Soft.*, 31(3):397–423, 2005.

[7] P. Lott, C. Woodward, and K. Evans. Algorithmically scalable block preconditioner for fully implicit shallow water equations in CAM-SE. *Comp. Geosci.*, 19:49–61, 2015. doi:10.1007/s10596-014-9447-6.

[8] K. Morris, M. N. L. D.W.I. Rouson, and S. Filippone. Exploring capabilities within ForTrilinos by solving the 3D burgers equation. *Sci. Programming*, 20(3):275–292, 2012.

[9] C. Newman and D. A. Knoll. Physics-based preconditioners for ocean simulation. *SIAM J. Sci. Comp.*, 35:S445–S464, 2013. doi:10.1137/120881397.

[10] M. Norman, , A. V. J. Larkin, and K. Evans. A case study of CUDA FORTRAN and OpenACC for an atmospheric climate kernel, implicit climate dynamics. *J. Comp. Sci.*, 9:1–6, 2015.