Experiences with Parallel Numerical Software Interoperability

William Gropp and Lois McInnes

In collaboration with Satish Balay, Steve Benson, Kris Buschelman, and Barry Smith

ARGONNE NATIONAL LABORATORY • UNIVERSITY OF CHICAGO •

Focus

- Highlight issues in building sharable component* software
  - ♦ Interfaces, performance, portability
- Discuss mistakes still often made in
  - ♦ designing,
  - ♦ maintaining,
  - ♦ documenting, and
  - ♦ testing components

* In this context we use the word *component* generically.

PETSc

- Initial project (1990):
  - Research into domain decomposition methods for parallel computers
  - Requires composing different methods
    - Preconditioners involving nested linear solvers, both direct and iterative
  - Requires performance and scalability
    - Performance requires flexibility in data structures, since this controls the performance of operations (load/store/flop)
    - Scalability requires attention to algorithms and to any sequential sections
- PETSc is not a complete framework
  - Hope and expect to interoperate with other tools
  - Success in
    - Legacy applications (PETSc works well with apps)
    - Other tools (PETSc plays well with peers)

Even Simple Components Can Be Difficult to Use

- BLAS
  - Missing operations like $y = x + \alpha y$
    - Focus on dense, not iterative, methods
  - Finding the libraries can be difficult
    - Resource discovery
- FFTs
  - Many not stateless
    - Convenient for many uses
    - Can cause *enormous* performance penalties where the user's use does not match this
- Vendor scientific routine libraries
  - Routine-specific data structures
  - Non-standard calling sequences (e.g., ESSL dgeev)
  - Resource discovery
    - Where is it? Is it licensed? Is it licensed for the parallel machine?

Interoperability

- Easier said than done
  - Concept mismatches
    - Different organization of operations
  - Control conflicts
    - At most one package can be in control
    - Common limitation is “user” or “app” code *or* package. The *Two-level* limitation: plays well with *only* one other
  - Performance conflicts
    - Different data structures hard to manage without copies
  - Tactical Details
    - Incompatible build or execute frameworks
      - Namespace conflicts
      - Assumes that all processes/threads participate in any operation
      - Whose makefile (or make) do I use?
      - Whose compilation script?
    - (See Curse of Orthogonality)

LANS

Case Studies

- Interfacing between PETSc and
  - ODE solvers
    - PVODE
  - Mesh management and discretization tools
    - SUMAA3d, SAMRAI, Overture
  - Linear algebra solvers
    - AMG, BlockSolve95, ILUTP, LUSOL, SPAI, SuperLU
  - Optimization software
    - TAO, Veltisto
  - Others
    - Matlab, ParMETIS

FLY through these

PETSc and PVODE

- PVODE
  - Parallel, robust, variable-order stiff and non-stiff ODE integrators
  - A. Hindmarsh et al. (LLNL)
  - http://www.llnl.gov/CASC/PVODE
  - L. Xu developed PVODE/PETSc interface
- Interface approach
  - PVODE
    - ODE integrator – evolves field variables in time
    - vector – holds field variables
    - preconditioner placeholder
  - PETSc
    - ODE integrator placeholder
    - vector
    - sparse matrix and preconditioner
- Comments
  - Interface between PETSc's TS component and PVODE required **no code alterations** in either package due to design allowing objects in one package to appear as objects in the other
  - Interface consists of **26 routines (873 lines of code)**
    - **1 new routine** for functionality not within PETSc: TSPVodeSetExactFinalTime()
    - Duplicate routines needed because PVODE implements its own Krylov solvers: TSPVodeGetIterations(), TSPVodeSetGMRESRestart(), TSPVodeSetLinearTolerance(), TSPVodeSetGramSchmidtType();
    - All other routines are simply wrappers to match APIs

PETSc and Mesh Management

- SUMAA3d
  - Scalable Unstructured Mesh Algorithms and Applications
  - L. Freitag (ANL), M. Jones (VA Tech), P. Plassmann (Penn State)
  - http://www.mcs.anl.gov/sumaa3d
  - L. Freitag and M. Jones developed SUMAA3d/PETSc interface
- SAMRAI
  - Structured adaptive mesh refinement
  - R. Hornung, S. Kohn (LLNL)
  - http://www.llnl.gov/CASC/SAMRAI
  - SAMRAI team developed SAMRAI/PETSc interface
- Overture
  - Structured composite meshes and discretizations
  - D. Brown, W. Henshaw, D. Quinlan (LLNL)
  - http://www.llnl.gov/CASC/Overture
  - K. Buschelman developed Overture/PETSc interface

SUMAA3d

- Interface Approach
  - SUMAA3d assembles vectors and matrices for use by PETSc's solvers
  - SUMAA3d creates a mapping between mesh unknowns and vector/matrix data in PETSc; then SUMAA3d uses this mapping to scatter vector data back onto the mesh (e.g., after a linear solve)
    - Uses PETSc "container" construct to attach SUMAA3d mapping to PETSc vector/matrix objects; the container construct had been **previously designed** to handle this type of situation
- Comments
  - Interface **required no alterations** to code in either package
  - Reference: "Mesh Component Design and Software Integration in SUMAA3d", L. Freitag, M. Jones, and P. Plassmann, SIAM Workshop on OO Methods for Interoperable Scientific and Engineering Computing, 1998

PETSc and Linear Solvers

- Interface Approach
  - ♦ Based on interfacing at the matrix level, where external linear solvers typically use a variant of compressed sparse row matrix storage
- AMG
  - ♦ Algebraic multigrid code by J. Ruge, K. Steuben, and R. Hempel (GMD)
  - ♦ http://www.mgnet.org/mgnet-codes-gmd.html
  - ♦ PETSc interface by D. Lahaye (K.U.Leuven), uses MatSeqAIJ
- BlockSolve95
  - ♦ Parallel, sparse ILU(0) for symmetric nonzero structure and ICC(0)
  - ♦ M. Jones (Virginia Tech.) and P. Plassmann (Penn State Univ.)
  - ♦ http://www.mcs.anl.gov/BlockSolve95
  - ♦ New code added to PETSc to support interface
    - Developed MatMPIRowbs matrix format
    - Incorporated PCPreSolve()/PCPostSolve() concepts to support preconditioner-specific actions needed before a linear solve (e.g., scaling and permuting)

PETSc and Linear Solvers (cont.)

- ILUTP
  - Drop tolerance ILU by Y. Saad (Univ. of Minnesota), in SPARSKIT
  - http://www.cs.umn.edu/~saad/
  - PETSc interface uses MatSeqAIJ
- LUSOL
  - Sparse LU, part of MINOS
  - M. Saunders (Stanford Univ)
  - http://www.sbsi-sol-optimize.com
  - PETSc interface by T. Munson (ANL), uses MatSeqAIJ
- SPAI
  - Sparse approximate inverse code by S. Barnhard (NASA Ames) and M. Grote (ETH Zurich)
  - http://www.sam.math.ethz.ch/~grote/spai
  - PETSc interface converts from any matrix format to SPAI matrix
- SuperLU
  - Parallel, sparse LU
  - J. Demmel, J. Gilbert, (U.C. Berkeley) and X. Li (NERSC)
  - http://www.nersc.gov/~xiaoye/SuperLU
  - PETSc interface uses MatSeqAIJ (currently only sequential interface supported); for parallel interface we could either copy matrix data from PETSc to SuperLU or develop a new matrix format in PETSc
    - Fairly mechanical to develop a new matrix class
    - Downside to a rich method set — raises a barrier to extensions
    - Need to automate much of this

PETSc and TAO

- TAO - Toolkit for Advanced Optimization
  - S. Benson, L. McInnes, and J. Moré
  - http://www.mcs.anl.gov/tao
- Initial TAO design uses PETSc for
  - Low-level system infrastructure - managing portability
  - Parallel linear algebra tools (SLES)
    - Veltisto (library for PDE-constrained optimization by G. Biros, Courant) – uses a similar interface approach
- TAO is evolving toward CCA-compliant component-based design
  - Support for ESI interfaces to various linear algebra libraries, e.g., Trilinos (M. Heroux, R. Lehoucq, with ESI interface by A. Williams)

PETSc and Others

- Matlab
  - PETSc socket interface to Matlab
    - Sends matrices and vectors to interactive Matlab session
  - PETSc interface to MatlabEngine
    - MatlabEngine – Matlab library that allows C/Fortran programmers to use Matlab functions in their programs
    - PetscMatlabEngine – unwraps PETSc vectors and matrices so that the MatlabEngine can understand them
      - PetscMatlabEngineCreate(), PetscMatlabEnginePutArray(), etc.
- ParMETIS
  - Parallel partitioning, G. Karypis (Univ. of MN)
  - http://www.cs.umn.edu/~karypis/metis/parmetis
  - PETSc interface from MatPartitioning component to ParMETIS
    - **Interface required 6 routines (191 lines of code)**
      - **Added 1 new routine**: MatPartitioningParmetisSetCoarseSequential();
      - All others are wrappers between APIs

Now It Gets Ugly
or
The Part You Have Been Waiting For

- “Just building the other package proved to be one of the larger headaches.”
- Portability: Header Files
- Portability: Standards
- Interoperation
- Other problems

Portability: Header Files

- Multiply defined (and inconsistent) preprocessor directives
  - Processed before namespaces
  - Both TAO and OOQP include an interface to PETSc vectors and both used
    #ifndef PETSCVECTOR_H
    #define PETSCVECTOR_H
    in their respective header files. Until we caught this problem only one of the two header files was being included.
- Header files that have prototypes for system routines that clash with the true system prototypes on some machines
  - See GNU autoconf recommendations
- Namespace conflicts.
  - The term 'Scalar' is defined in multiple packages.
- Missing “extern C” in C headers impedes use with C++.

Portability: Standards

- Use of compiler switches to change the *language*
  - -r8 for some Fortran compilers
- Aggressive use of bleeding edge features
  - (Changes with time as compilers mature)
  - (sophisticated) Templates in C++
  - ISO CPP features (like –Dmalloc=malloc)
- Undocumented use of runtime extensions
  - Nonstandard timer calls deep in the bowls of the software
- Use of hand-entered floating point machine constants which are entered as unions of int and float, and entered as int, incorrectly

Interoperation

- Software that requires argc and argv deep inside the libraries, as opposed to within an initialization routine
  - PetscGetArgs — allows PETSc to provide other applications with access the command line
  - Added to help PETSc play well with others
- Deletion of objects. When objects are embedded, or pointed to, from within other objects, it is not always clear who should destroy each of the objects. Double deletion and memory leaks are a problem.
  - Requires special care for objects shared among components
  - Automate leak and dangling reference detection (e.g., unreclaimed at exit)
  - PETSc (and MPICH) includes tracing and overwrite-checking malloc/calloc/strdup/free routines

Interoperation con’t

- IO that cannot be turned off
  - ♦ Redirection of IO just barely adequate
- Routines that do not return error conditions
  - ♦ Always return success or
  - ♦ Exit on error (!) or
  - ♦ Print to stdout (not even stderr!)
- Software that provides no point of entry for other tools
  - ♦ E.g., LU factorization routines that are hardwired to use their provided preordering routines, even though all they really need is the permutations
    - Lack of modularity in the package
    - Did not plan to interoperate
    - *Components must be prepared to give up some control*

Interoperation: MPI

- Packages built with different MPI implementations cannot interoperate
  - MPI does not specifiy the exact form of basic objects (e.g., MPI_Comm) or constants (e.g., MPI_COMM_WORLD)
    - Allows implementer maximum freedom to be clever:
      - Datatype size in bytes encoded in datatype
      - Instance-specific error message in MPI error code
    - Each implementation's mpi.h file is different
- Possible fixes
  - Generic MPI that translates between a defined MPI and any other, exploiting the PMPI interface.
    - Pro: does everything, requires no changes to other codes
    - Con: adds a layer; uses up PMPI
  - "Constant free" MPI works for MPI's with same-sized types but different values
    - Pro: directly call MPI; retain PMPI for other tools
    - Con: small changes in libraries required (but only when compiling for this MPI "implementation")

Common Problems that Hinder Software Interoperability

Assuming global control of a simulation rather than being a peer component

- Software that assumes it will always be used at the highest level of control of a simulation
- Software that assumes it will be used for just 1 activity at a time
  - E.g., linear algebra tools that use "secret" globals to store matrices etc. so that only one solver may be active at a time
  - This approach is suitable for a high-level, easy-to-use interface, but should **never** be part of the basic package design
- Software that requires that it initializes MPI instead of allowing the user (or another component) to do so.
  - Note that MPI has routines designed for *exactly* this situation.
  - MPI-2 allows NULL for &argc, &argv.

Mistakes Still Being Made in 2001

- Ignorance of standards
- Failure of components to exceed lifetime of consumer applications
- Requiring the component to be the master
- Printing error messages and/or exiting program
- Mandating interactive input
- Makefiles for particular systems
- Lack of portability in general
- Poor documentation
- Poor testing
- Poor examples
- Name space pollution
- Monolithic library

- Avoid the curse of orthogonality
  - ♦ Concepts should be orthogonal
  - ♦ Presentation (methods/routines) should not

GNU Autotools
Bill's View

- Autoconf
  - Good enough. Knows a great deal about portability misfeatures in many systems. 2.13 has some Fortran and C++ support
  - GNU-centrism can be overridden
  - Documentation is ghastly
- Automake
  - Too GNU centric. Perfect for GNU developers, problematic for others
    - File dependency generator requires GNU development environment
    - Generated files are fragile because faulty file system time stamps can cause make to attempt to rebuild them
    - Rebuild targets are not correct for user system (expects particular versions of autoconf and automake; doesn't package extension macros)
- Libtool
  - Almost but not quite
  - Excellent resource for wildly (and pointlessly) varying shared library tools and command line parameters
  - Significantly perturbs the development environment, particularly for parallel applications