

Mathematical and math software issues in the SWIM FSP project

David Keyes

Columbia University

SciDAC TOPS project lead

From DBB's seven questions

- 1) **Specify the functionality for the component**
- 2) **Determine the interface to provide this functionality**
- 3) **Identify code-specific data**
- [...]
- 6) **Physics or mathematical analysis/development needs**
 - opportunities for improvement**
 - additional opportunities for parallelism that are not presently realized**
- 7) **Computational needs this component will have when used for the fast MHD campaign**
 - computations**
 - memory requirements**
 - number of calls or total time to carry out its designated task in a typical IPS simulation**
 - kind of parallelism (threaded or distributed memory or both)**



The TOPS solver component

- **The TSC is briefly described herein**
 - **5 slides from our 30-slide SC'05 announcement**
 - **URL to more documentation**
- **Main focus of this talk is more on identifying abstract mathematical issues that should be collaboratively explored**



Why TOPS should collaborate with SWIM

- **Leverage the TOPS project for fusion simulations**
- **Enhance the relevance of the TOPS project by means of its application of magnetically confined fusion**
- **Identify mathematical research issues for doctoral students throughout TOPS academic institutions**



David Keyes
wants to impose
the TOPS Solver
Component on the
SWIM project

Don't believe
everything that
Randall tells
you about the
TOPS project.



“In TOPS, we have principles!

And if you don't like them ...

We have others!”

(with apologies to Groucho Marx)



TOPS: SciDAC's scalable solvers project

One of three algorithmic integrated software infrastructure centers (ISICs) in the Scientific Discovery through Advanced Computing (SciDAC) initiative



Columbia



Courant



Old Dominion

Carnegie Mellon



Colorado



Tennessee

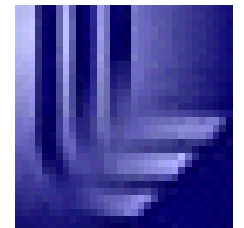
Berkeley
University of California



Argonne



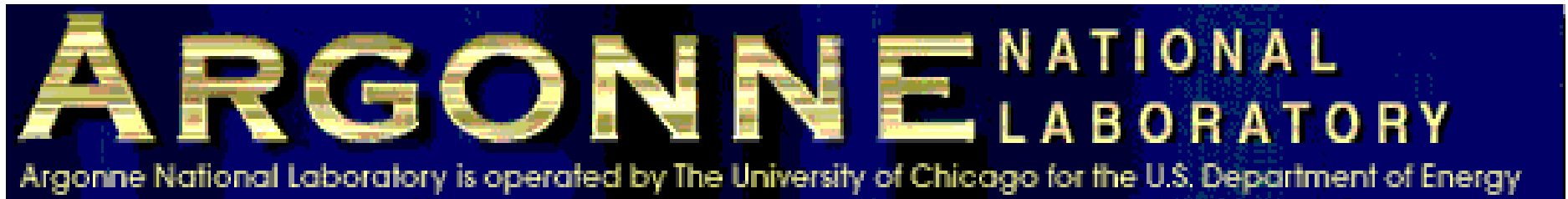
Lawrence Berkeley



Lawrence Livermore



Who we are in TOPS...



... the PETSc and TAO people



... the hypre and Sundials people



Berkeley Lab

... the SuperLU and PARPACK people

We are one of three math ISICs

APDEC

Applied Partial Differential Equations Center,
directed by P. Colella

TSTT

Terascale Simulation Tools & Technologies,
directed by D. Brown, L. Diachin, J. Glimm



Terascale Optimal PDE Simulations
directed by yours truly



Scope for TOPS

- Design and implementation of “solvers”

- Time integrators
(w/ sens. anal.)

$$f(\dot{x}, x, t, p) = 0$$

- Nonlinear solvers
(w/ sens. anal.)

$$F(x, p) = 0$$

- Optimizers

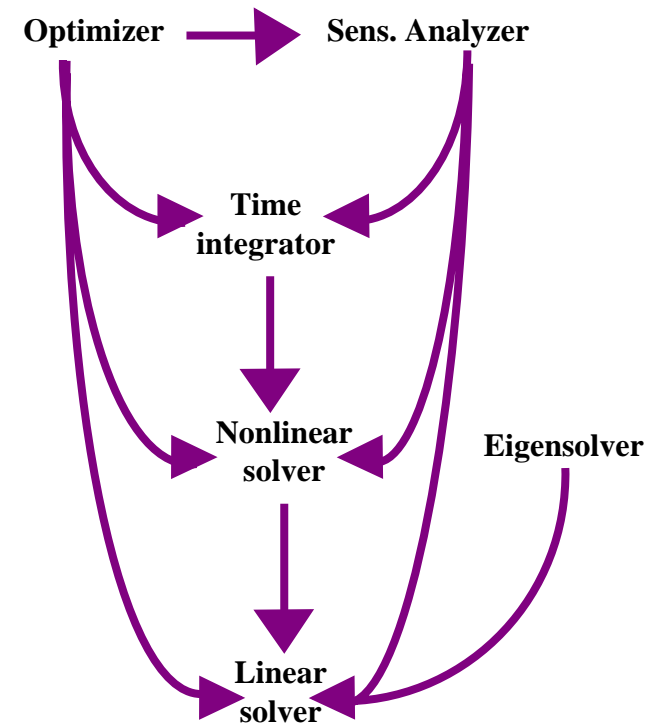
$$\min_u \phi(x, u) \text{ s.t. } F(x, u) = 0, u \geq 0$$

- Linear solvers

$$Ax = b$$

- Eigensolvers

$$Ax = \lambda Bx$$



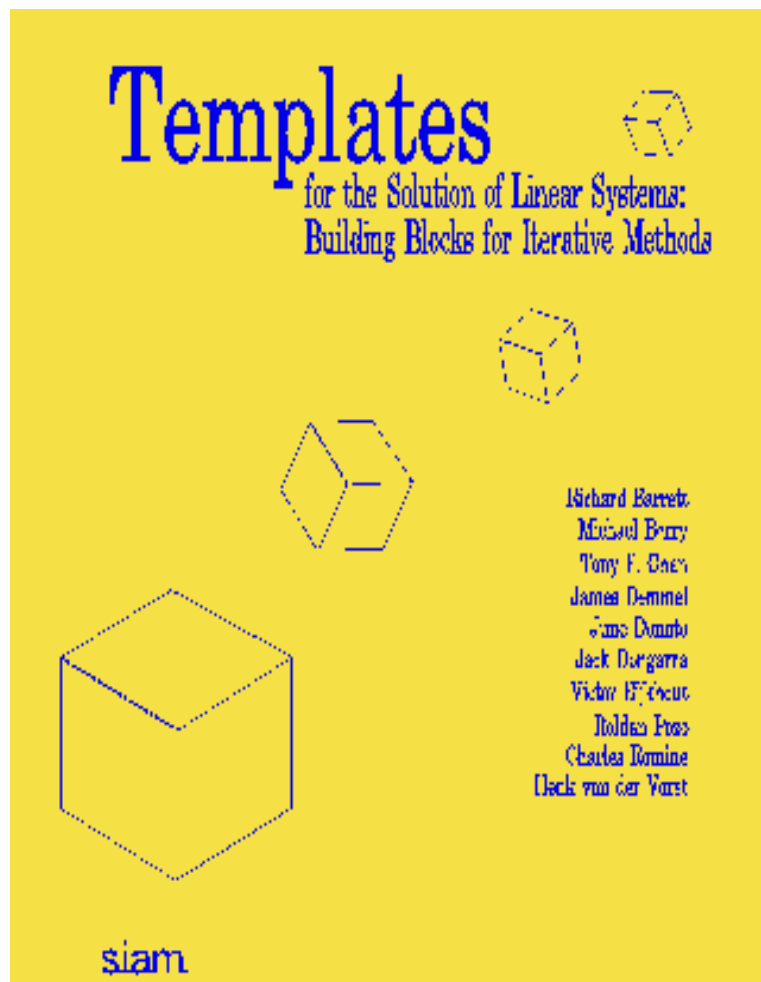
→ Indicates dependence

- Software integration
- Performance optimization



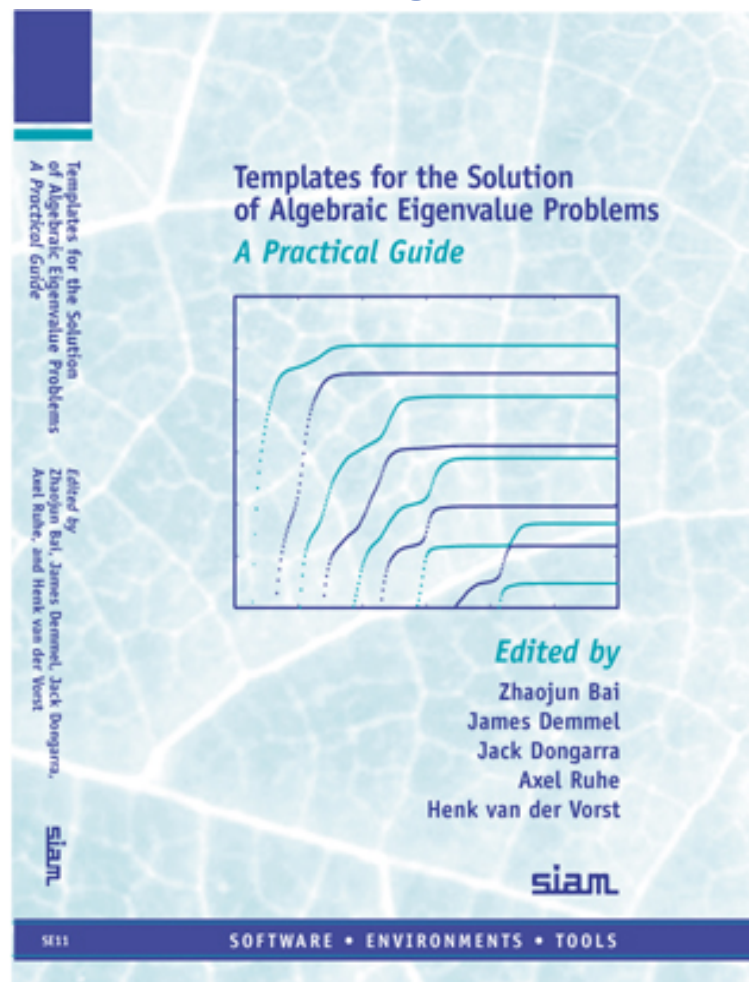
TOPS precursors

www.netlib.org/templates



124 pp.

www.netlib.org/etemplates



410 pp.

... these are good starts, but not adequate for SciDAC scales!



TOPS is interested in internals

$$Ax = b$$

AORSA, M3D, Nimrod, TORIC, ...

$$Ax = \lambda Bx$$

Nimrod

$$F(x, p) = 0$$

TSC and others during project scope of SWIM

$$f(\dot{x}, x, t, p) = 0$$

TSC, nonlinear MHD, anything that can be formulated as method-of-lines, maybe *all*

$$\min_u \phi(x, u) \text{ s.t. } F(x, u) = 0, u \geq 0$$

the future beyond first 3-year SWIM scope



Challenges for math software designers

- **The past challenge:**

Increase functionality and capability for a small number of users who are expert in the domain of the software

- **A future challenge:**

Increase ease of use (for correctness *and* for efficiency) for a large number of users who are expert in something else



Design principle: multiple layers

- **Top layer (all users)**
 - **Abstract interface featuring language of application domain, hiding details, with conservative parameter defaults**
 - *Robustness, correctness, ease of use*
- **Middle layers (experienced users)**
 - **Rich collection of state-of-the-art methods and data structures, exposed upon demand, highly configurable**
 - *Capability, algorithmic efficiency, extensibility, composability, comprehensibility of performance and resource use*
- **Bottom layer (developers)**
 - **Support for variety of execution environments**
 - *Portability, implementation efficiency*



What physicists want in math software

- **Develop code “without having to make bets”**
 - accomplish certain abstract mathematical tasks
 - stay agnostic about particular solution methods and codes
 - run on laptops (on travel), low-cost unmetered clusters (at work), and on unique shared national resources
- **Ordered goals (need them all for production use)**
 - usability and robustness
 - portability
 - algorithmic efficiency (optimality) and implementation efficiency (within a processor and in parallel)
- **Algorithmic optimality and software stability**
 - scalable methods needed for multiscale problems
 - no “hand coding” for evanescent environments



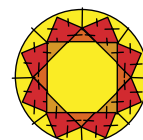
Applied math issues in SWIM

- **Most are common to general simulation efforts:**
 - modeling, formulation, discretization
 - solution/integration, error estimation, adaptation
 - convergence analysis, performance optimization
- **Our principal interest in SWIM is:**
 - accommodating discretizations that are higher order in time and space (for algorithmic and architectural efficiency)
 - greater implicitness, linearly and nonlinearly
- **The multiphysics coupling focus of the FSP introduces some less frequently explored issues, as well**
 - combinations of many types of physics, previously not studied together
 - more physics means more interactions that could lead to linear or nonlinear waves and further restrict CFL
 - implicitness desirable to avoid instability, and also to be able to get beyond first-order in operator-split formulations



TOPS solver components

- **Language-independent software components for the scalable solution of large linear and nonlinear algebraic systems arising from either structured or unstructured meshes**
 - Solve $Ax=b$
 - Solve $f(u)=0$, where $f:R^n \rightarrow R^n$
- **Compliant with the Common Component Architecture (CCA)**
 - Interfaces use SIDL (Scientific Interface Definition Language) and the Babel (LLNL) language interoperability tool
 - Common interfaces to facilitate easy experimentation with different solvers developed by different institutions
- **Developed by the TOPS and CCTTSS (CCA) SciDAC projects**



CCA
Common Component Architecture



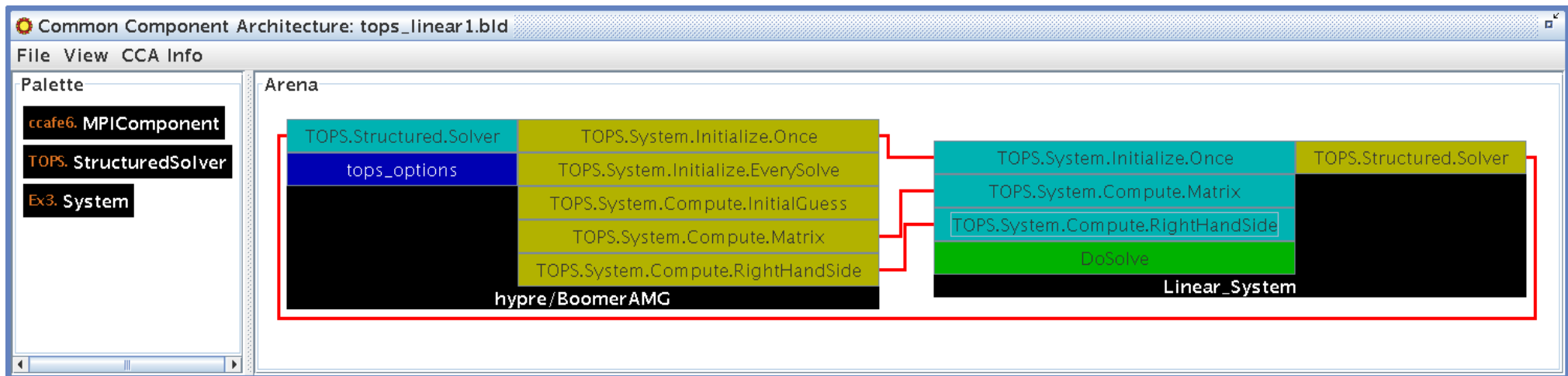
Using TOPS solver components

- The application developer constructs a CCA component that implements the TOPS.System interfaces. This component defines the algebraic system to be solved and calls for its solution.
- The solver and the TOPS.System component can be combined using one of:
 - a traditional programming language
 - a component scripting language
 - a component GUI such as provided by the Ccaffeine framework (SNL) ... see examples in following slides
- The solver and TOPS.System component then collaborate to solve one or more algebraic problems. Complex applications will likely also couple several additional CCA components.



TOPS linear solver components

- The application *provides* ports for
 - Initializing the linear system
 - Computing the coefficient matrix, A
 - Computing the right-hand-side vector, b
- The application *uses* a TOPS solver port

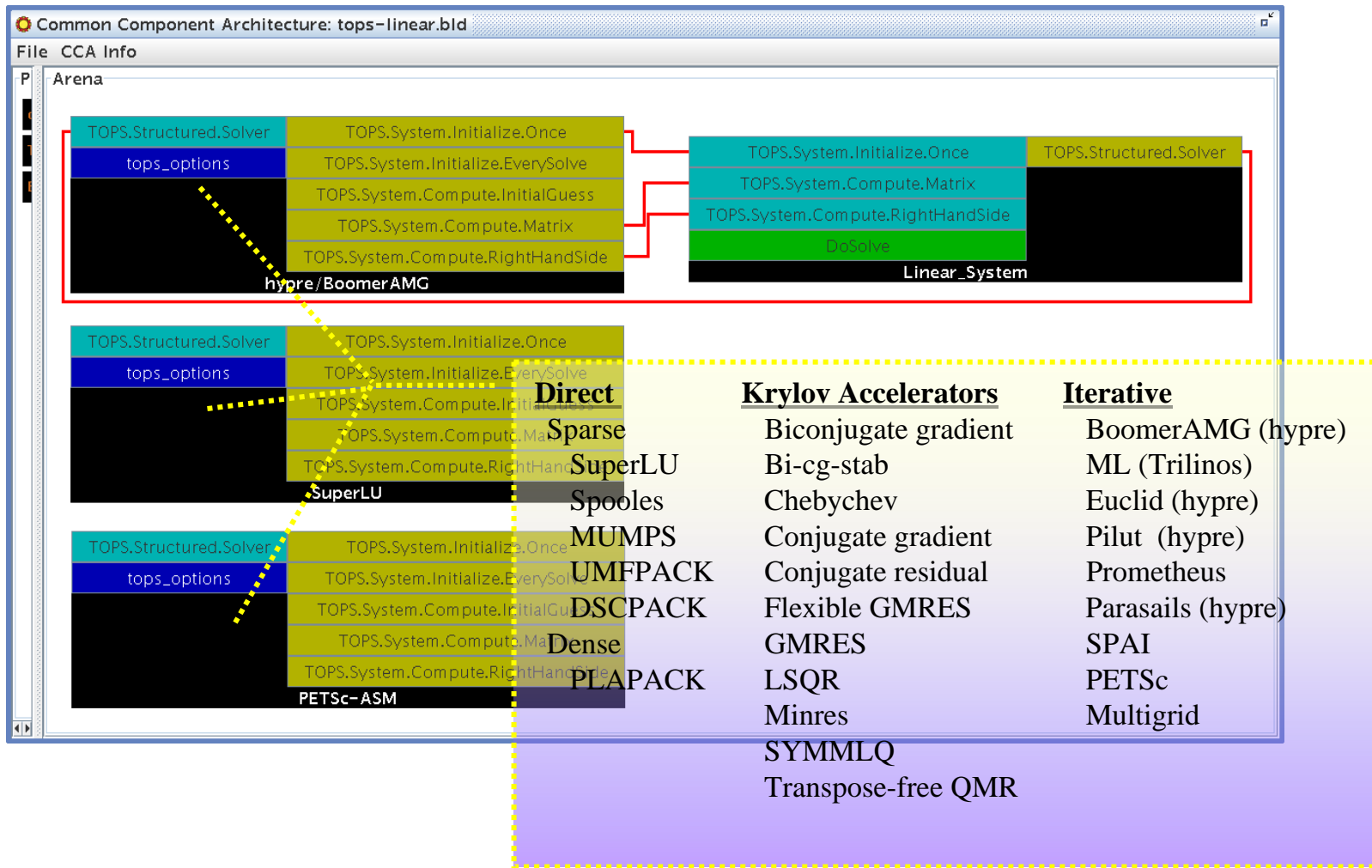


This wiring diagram using the GUI within the Caffeine framework (SNL) depicts solving a linear system arising from a structured mesh problem. The BoomerAMG solver within hypre (LLNL) is employed. The large black boxes represent components. The small light blue boxes denote *provides ports* (functionality provided for use by another component); the small gold boxes denote *uses ports* (functionality needed from another component);



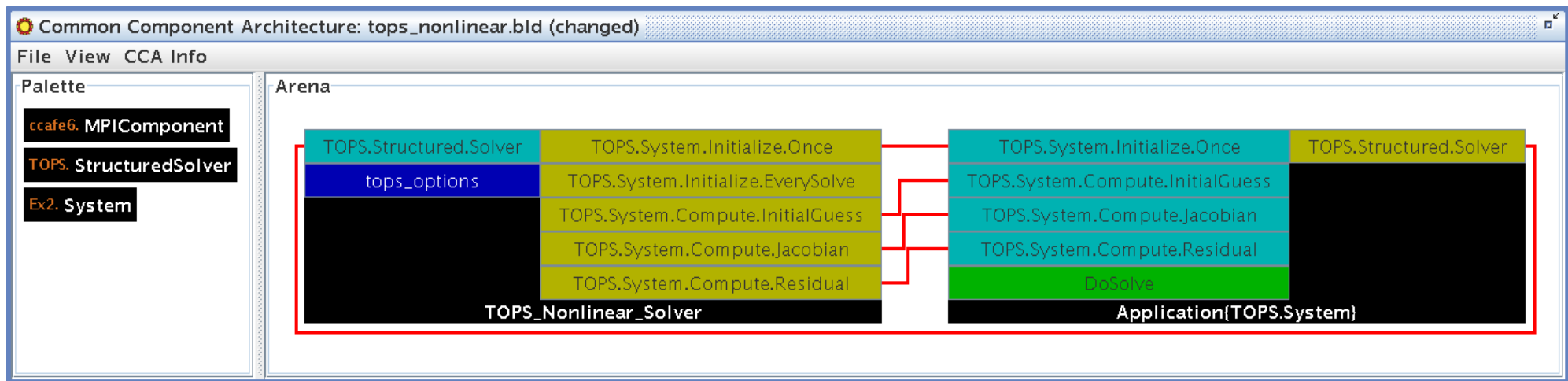
Facilitating experimentation

TOPS solvers enable applications scientists to easily experiment with vast numbers of linear solvers without needing to make *a priori* (compile time) choices about data structures and algorithms.



TOPS nonlinear solver components

- The TOPS design makes natural the transition from linear to nonlinear problems
- The application *provides* ports for
 - Initializing the nonlinear system
 - Computing the residual vector, $f(u)$
 - Computing the associated Jacobian matrix, $f'(u)$ (optional)
- The application *uses* a TOPS solver port

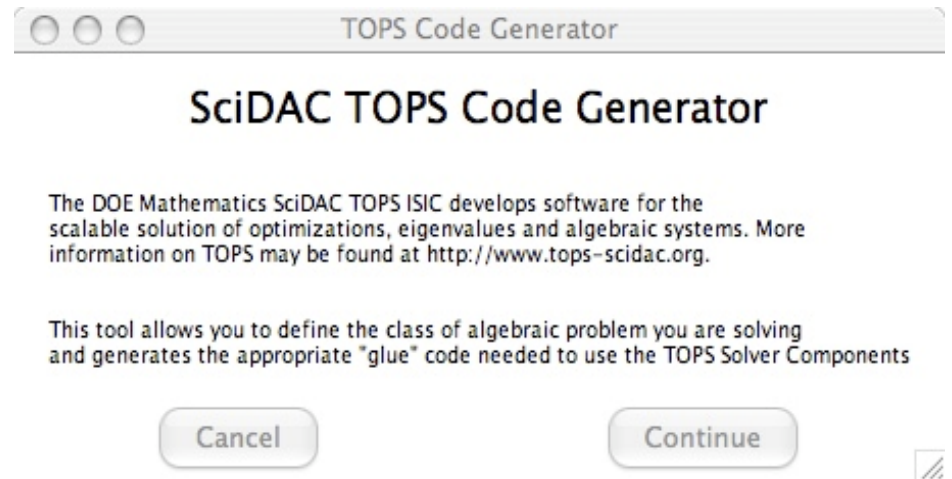


This wiring diagram depicts solving a nonlinear system arising from a structured mesh problem using a Newton-based nonlinear solver in PETSc (ANL).

Auxiliary TOPS tools

- TOPS Software Installer:
Can download and install a large variety of solver packages.
 - Eases the difficulties of manual installation of the various underlying toolkits
 - Ensures that the toolkits are all built with the same compilers and compiler options

- TOPS Component Generator:
Can generate the SIDL for your problem and all the boilerplate code needed to use it as a CCA component
 - Essentially, applications simply need to provide source code for the methods that define an algebraic linear or nonlinear problem.



Current status and future plans

- **The complete source code for TOPS components, along with the TOPS Software Installer, are available:**

<http://www.mcs.anl.gov/scidac-tops/solver-components/tops.html>

- **Future work includes:**
 - **Collaborate with early adopters of TOPS components**
 - **Explore the adaptivity of TOPS solver parameters and algorithms in response to changing conditions during long-running simulations, as an application of machine learning techniques to scientific computing**



From DBB's seven questions

1) Specify the functionality for the component

- *all five solver types implemented for various problem structures*
- *linear solvers componentized*

2) Determine the interface to provide this functionality

3) Identify code-specific data

[...]

6) Physics or mathematical analysis/development needs

opportunities for improvement

- *tune optimal complexity solvers to existing implicit formulations*
- *combine existing physics codes into more implicit couplings*

additional opportunities for parallelism that are not presently realized

- *sometimes the solver is the bottleneck; sometimes the bottleneck is elsewhere (e.g., two separate physics phases that cannot be simultaneously load-balanced)*



From DBB's seven questions

7) Computational needs this component will have when used for the fast MHD campaign

computational complexity

- ideally, $O(N \log N)$ for sparse linear and nonlinear solves

memory requirements

- ideally, $O(N)$ for sparse linear and nonlinear solves

number of calls or total time to carry out its designated task in a typical IPS simulation

- one call per “solve” and many call-backs to application “provides” ports

kind of parallelism (threaded or distributed memory or both)

- MPI is universal; we do not depend on threads

- smoothest path towards multithreads/multicore is underneath MPI



Strategies for Implicit Code Coupling

David Keyes

Columbia University

SciDAC TOPS project lead

The FSP is about code coupling

- **Many combinations exist**
 - Particle-based and field-based code pair
 - Pair of field-based codes
 - Codes that share a common domain of definition (though likely different grids)
 - Codes whose domains of definition share only an interface across which fluxes pass
 - Codes in steady state or that time-march together
 - Time-marching code paired with quasi-steady-state code
- **Abstract discussion here can pertain to all**
 - Generalizes to more than two such codes



Algebraic template for implicit code coupling

- Consider system $F(u) = 0$ partitioned by physics as

$$\begin{cases} F_1(u_1, u_2) = 0 \\ F_2(u_1, u_2) = 0 \end{cases}$$

- Assume that F_1 can be solved for u_1 , given u_2 as a parameter vector, resp. for F_2 and u_2 , given u_1
 - Parameter vector may represent source term, boundary data, coefficients of the operator
 - Prior to coupling, parameters have simply been assumed and fixed
 - FSP agenda is to build more realistic model by satisfying mutual consistency
 - “Triangular case” in which either $\frac{\partial F_1}{\partial u_2} = 0$ or $\frac{\partial F_2}{\partial u_1} = 0$ is not very interesting



Example of such a system

- Consider a pair of evolution equations

$$\begin{cases} \frac{\partial u_1}{\partial t} + f_1(u_1, u_2) = 0 \\ \frac{\partial u_2}{\partial t} + f_2(u_1, u_2) = 0 \end{cases}$$

- f_1 and f_2 contain all spatial derivatives
- Discretize implicitly in time, e.g., to second order

$$\frac{\partial u_i}{\partial t}(t_k) \approx \frac{3u_{i,k} - 4u_{i,k-1} + u_{i,k-2}}{2\Delta t}$$

- Bury dependence on past iterates in the subscript k in F_k , initialize, and write system at each step as

$$F_{i,k}(u_{1,k}, u_{2,k}) = 0, i = 2, 3, \dots$$



Multiphysics coupling (1): nonlinear GS

- Use superscript for iterations within a timestep and suppress subscript where understood
- Given previous step values $\{u_{1,k-1}, u_{2,k-1}\}$
- Define initial iterate $\{v^0, w^0\} = \{u_{1,k-1}, u_{2,k-1}\}$
- For $\ell=1, 2, \dots$, until convergence, do
 - Solve for v^ℓ in $F_1(v^\ell, w^{\ell-1}) = 0$
 - Solve for w^ℓ in $F_2(v^\ell, w^\ell) = 0$
- Then current step values are $\{u_{1,k}, u_{2,k}\} = \{v^\infty, w^\infty\}$
- Nonlinear block Jacobi version is also relevant
- *This is generally slowly convergent*



Multiphysics coupling (2): partial elimination

- Can formally solve for u_1 in $F_1(u_1, u_2) = 0$

$$u_1 \equiv G(u_2)$$

- Then second equation is $F_2(G(u_2), u_2) = 0$

- Jacobian

$$\frac{dF_2}{du_2} = \frac{\partial F_2}{\partial u_1} \frac{\partial G}{\partial u_2} + \frac{\partial F_2}{\partial u_2}$$

can be applied to a vector in matrix-free manner,
without knowing explicit functional form of $G(\cdot)$ to
solve second equation

- *This is generally very expensive per iteration*



Multiphysics coupling (3): nonlinear Schwarz

- Given initial iterate $\{u_1^0, u_2^0\}$
- For $\ell=1, 2, \dots$, until convergence, do
 - Define $G_1(u_1, u_2) \equiv \delta u_1$ by $F_1(u_1^{\ell-1} + \delta u_1, u_2^{\ell-1}) = 0$
 - Define $G_2(u_1, u_2) \equiv \delta u_2$ by $F_2(u_1^{\ell-1}, u_2^{\ell-1} + \delta u_2) = 0$
 - Then solve
$$\begin{cases} G_1(u_1, u_2) = 0 \\ G_2(u_1, u_2) = 0 \end{cases}$$
- Jacobian:
$$\begin{bmatrix} \frac{\partial G_1}{\partial u_1} & \frac{\partial G_1}{\partial u_2} \\ \frac{\partial G_2}{\partial u_1} & \frac{\partial G_2}{\partial u_2} \end{bmatrix} \approx \begin{bmatrix} I & \left(\frac{\partial F_1}{\partial u_1}\right)^{-1} \frac{\partial F_1}{\partial u_2} \\ \left(\frac{\partial F_2}{\partial u_2}\right)^{-1} \frac{\partial F_2}{\partial u_1} & I \end{bmatrix}$$



Reusing legacy code

- To apply Newton's method block-by-block as needed in previous three slides, it appears that we need

$$\frac{\partial F_i}{\partial u_j}, j = i$$

- In fact, we need only solve systems with these Jacobian diagonal blocks, which can be done matrix-free using a Krylov method, provided:
 - we can easily apply the forward action of these blocks to the appropriate subvector
 - we can effectively precondition these blocks
- These two types of vector-to-vector maps can often be carried out with legacy code



Applying Jacobian block to a vector

- To form a matrix-vector product with the Jacobian block, we need to call the legacy code residual evaluation subroutine one additional time with a perturbed argument

$$\frac{\partial F_i}{\partial u_j} v \approx \frac{1}{\varepsilon} [F_i(\dots, u_j + \varepsilon v, \dots) - F_i(\dots, u_j, \dots)]$$

- First-order directional differencing is shown here; higher-order is also possible at the cost of additional residual evaluations



“Inverting” Jacobian block on a vector

- Most legacy codes have a vector-to-vector map that produces an update to the state vector, given an evaluation of the current residual

$$\mathcal{M}_i : F_i(\dots, u_i, \dots) \mapsto \delta u_i$$

- This map is likely to be an excellent preconditioner for the Jacobian, since is functionally the (opposite of the) action of the Jacobian inverse, in Newton’s method

$$\delta u_i = - \left(\frac{\partial F_i}{\partial u_i} \right)^{-1} F_i(u_i)$$



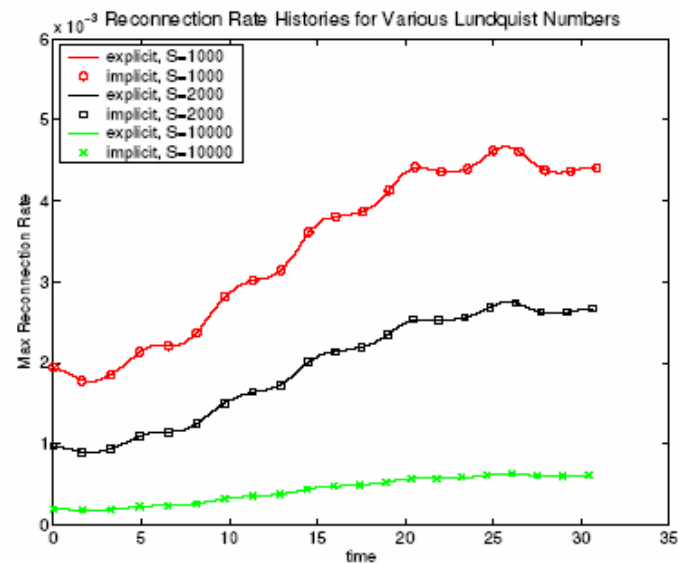
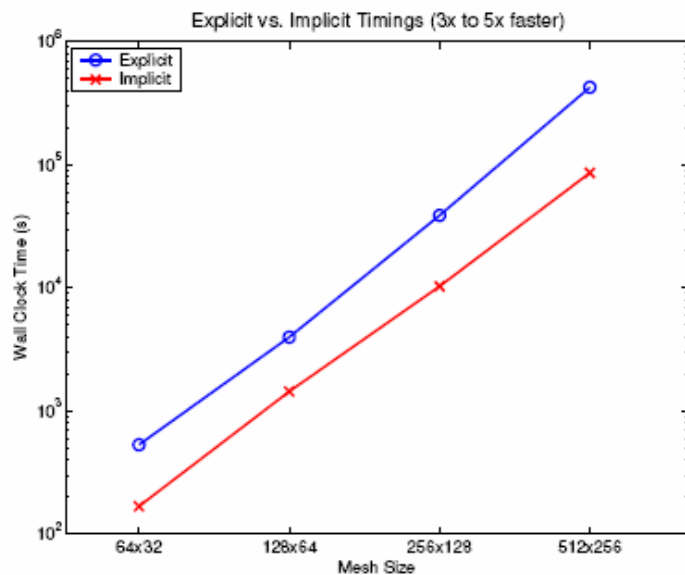
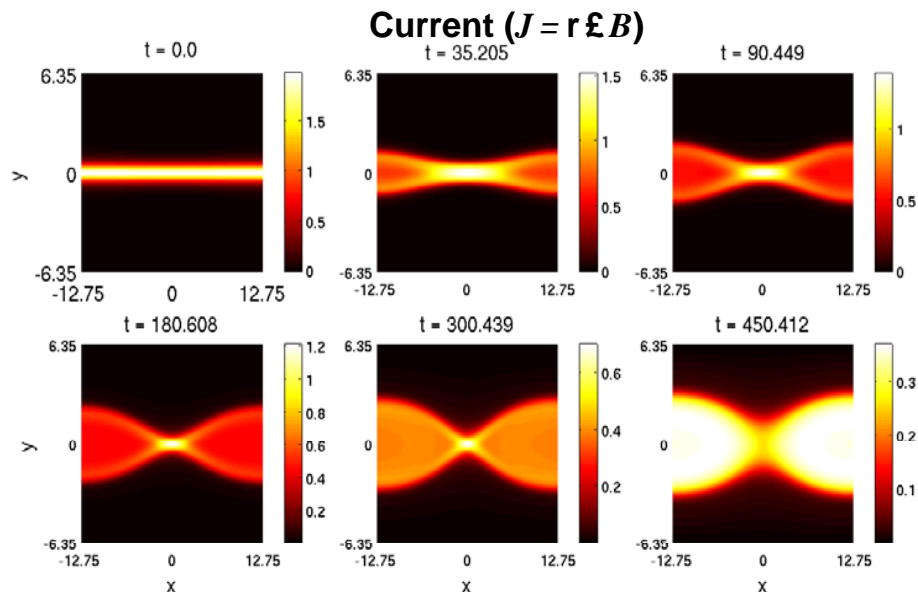
Implicit code coupling conclusion

- We conclude that any pair of codes, each of which already efficiently solves for its own principal state vector, given the ability to evaluate its own residual, can be coupled into an overall outer Newton's method code harness, with a stacked state vector and block Jacobian operations
- A versatile solver toolkit is needed to supply algorithmic pieces (Newton with trust region or back tracking, Krylov, possibly other preconditioners, etc.)
- The solver toolkit may want to build its own distributed state vector object to offer to the legacy codes, or register theirs
- Development and demonstration of this capability can be done for the FSPs using TOPS software
- *NB: this Jacobian-free Newton enhancement can also be provided for an individual code*



Nonlinear implicit version of GEM MRC code

- Newton-Krylov wrapped around existing explicit formulation
- Time-step increased order of magnitude or more beyond CFL limit
- Nonlinear consistency on each timestep



J. Brin et al., "Geospace Environmental Modeling (GEM) magnetic reconnection challenge," *J. Geophys. Res.* 106 (2001) 3715-3719.



Newton-Krylov-Schwarz: a PDE applications “workhorse”

$$F(u) \approx F(u_c) + F'(u_c)\delta u = 0$$

$$u = u_c + \lambda \delta u$$

$$J\delta u = -F$$

$$\delta u = \underset{x \in V \equiv \{F, JF, J^2F, \dots\}}{\operatorname{argmin}} \{Jx + F\}$$

$$M^{-1}J\delta u = -M^{-1}F$$

$$M^{-1} = \sum_i R_i^T (R_i J R_i^T)^{-1} R_i$$



Newton

nonlinear solver

asymptotically quadratic



Krylov

accelerator

spectrally adaptive



Schwarz

preconditioner

parallelizable



Philosophy of Jacobian-free N-K

- To *evaluate* the linear residual, we use the true $F'(u)$, giving a true Newton step and asymptotic quadratic Newton convergence
- To *precondition* the linear residual, we do anything convenient that uses understanding of the dominant physics/mathematics in the system and respects the limitations of the parallel computer architecture and the cost of various operations:
 - Jacobian blocks decomposed for parallelism (Schwarz)
 - Jacobian of lower-order discretization
 - Jacobian with “lagged” values for expensive terms
 - Jacobian stored in lower floating point precision
 - Jacobian of simpler, related discretization
 - preconditioner from operator-splitting, or legacy code map



Philosophy of Jacobian-free NK, cont.

- These motivations are not new; most large-scale application codes *also* take “short cuts” on the approximate linearization to be inverted – based on physical intuition
- The problem with many codes is that they do not anywhere have an accurate global Jacobian operator; they use *only* this approximate linearization
- This leads to a weakly nonlinearly converging “defect correction method”

- Defect correction:

$$B \delta u^k = -F(u^k)$$

- in contrast to preconditioned Newton:

$$B^{-1} J(u^k) \delta u^k = -B^{-1} F(u^k)$$



TOPS software was designed for this

- **High performance, massively parallel**
- **Has been behind Gordon Bell prizes in 1999, 2003, 2004**
- **Object-oriented, composable, extensible**
- **TOPS has supported five fusion codes so far under SciDAC (M3D, Nimrod, AORSA, GTC, prototype NKS GEM solver)**



Follow-up

- **Nonlinear Schwarz**
Cai & Keyes, SIAM J Sci Comput (2000)
- **Jacobian-free Newton-Krylov**
Knoll & Keyes, J Comp Phys (2004)
- **TOPS project**
<http://www.tops-scidac.org>
- **TOPS solver components**
<http://www-unix.mcs.anl.gov/scidac-tops/solver-components/tops.html>

