The DOE ACTS Collection

How can it be used?

UC Berkeley - CS267 - March 31, 2004

Tony Drummond
Lawrence Berkeley National Laboratory
acts-support@nersc.gov
ladrummond@lbl.gov
Motivation - Why do we need software libraries?

Large Scientific Codes: A Common Programming Practice

- Algorithmic Implementations
- Application Data Layout
- Tuned and machine Dependent modules
- Control
- I/O
Motivation - Why do we need software libraries?

New Architecture:
• May or may not need re-rewriting

New Developments:
• Difficult to compare

New Architecture:
• Extensive re-rewriting

New or extended Physics:
• Extensive re-rewriting or increase overhead

New Architecture:
• Minimal to Extensive rewriting

New Architecture or S/W:
• Extensive tuning
• May require new programming paradigms
• Difficult to maintained!
Motivation - Why do we need software libraries?

An Alternative Approach

USER's APPLICATION CODE
(Main Control)

Available Libraries & Packages
Application Data Layout

Available Libraries & Packages
Algorithmic Implementations

Available Libraries
I/O

Tuned and machine Dependent modules
Shortcomings?

"We need to move away from a coding style suited for serial machines, where every macrostep of an algorithm needs to be thought about and explicitly coded, to a higher-level style, where the compiler and library tools take care of the details. And the remarkable thing is, if we adopt this higher-level approach right now, even on today's machines, we will see immediate benefits in our productivity."

*Numerical Recipes: Does This Paradigm Have a future?*
## ACTS Tools

<table>
<thead>
<tr>
<th>Category</th>
<th>Tool</th>
<th>Functionalities</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Numerical</strong></td>
<td>Aztec</td>
<td>Algorithms for the iterative solution of large sparse linear systems.</td>
</tr>
<tr>
<td></td>
<td>Hypre</td>
<td>Algorithms for the iterative solution of large sparse linear systems, intuitive grid-centric interfaces, and dynamic configuration of parameters.</td>
</tr>
<tr>
<td></td>
<td>PETSc</td>
<td>Tools for the solution of PDEs that require solving large-scale, sparse linear and nonlinear systems of equations.</td>
</tr>
<tr>
<td></td>
<td>OPT++</td>
<td>Object-oriented nonlinear optimization package.</td>
</tr>
<tr>
<td></td>
<td>SUNDIALS</td>
<td>Solvers for the solution of systems of ordinary differential equations, nonlinear algebraic equations, and differential-algebraic equations.</td>
</tr>
<tr>
<td></td>
<td>ScaLAPACK</td>
<td>Library of high performance dense linear algebra routines for distributed-memory message-passing.</td>
</tr>
<tr>
<td></td>
<td>SuperLU</td>
<td>General-purpose library for the direct solution of large, sparse, nonsymmetric systems of linear equations.</td>
</tr>
<tr>
<td></td>
<td>TAO</td>
<td>Large-scale optimization software, including nonlinear least squares, unconstrained minimization, bound constrained optimization, and general nonlinear optimization.</td>
</tr>
<tr>
<td><strong>Code Development</strong></td>
<td>Global Arrays</td>
<td>Library for writing parallel programs that use large arrays distributed across processing nodes and that offers a shared-memory view of distributed arrays.</td>
</tr>
<tr>
<td></td>
<td>Overture</td>
<td>Object-Oriented tools for solving computational fluid dynamics and combustion problems in complex geometries.</td>
</tr>
<tr>
<td><strong>Code Execution</strong></td>
<td>CUMULVS</td>
<td>Framework that enables programmers to incorporate fault-tolerance, interactive visualization and computational steering into existing parallel programs.</td>
</tr>
<tr>
<td></td>
<td>Globus</td>
<td>Services for the creation of computational Grids and tools with which applications can be developed to access the Grid.</td>
</tr>
<tr>
<td></td>
<td>PAWS</td>
<td>Framework for coupling parallel applications within a component-like model.</td>
</tr>
<tr>
<td></td>
<td>SILOON</td>
<td>Tools and run-time support for building easy-to-use external interfaces to existing numerical codes.</td>
</tr>
<tr>
<td></td>
<td>TAU</td>
<td>Set of tools for analyzing the performance of C, C++, Fortran and Java programs.</td>
</tr>
<tr>
<td><strong>Library Development</strong></td>
<td>ATLAS and PHIPAC</td>
<td>Tools for the automatic generation of optimized numerical software for modern computer architectures and compilers.</td>
</tr>
<tr>
<td></td>
<td>PETE</td>
<td>Extensible implementation of the expression template technique (C++ technique for passing expressions as function arguments).</td>
</tr>
</tbody>
</table>
CALL BLACS_GET( -1, 0, ICTXT )
CALL BLACS_GRIDINIT( ICTXT, 'Row-major', NPROW, NPCOL )
CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )
CALL PDGESV( N, NRHS, A, IA, JA, DESCA, IPIV, B, IB, JB, DESCB, INFO )

- **ksp_type** [cg,gmres,bcgts,tfqmr,...]
- **pc_type** [lu,ilu,jacobi,sor,asm,...]

More advanced:
- **ksp_max_it** <max_iters>
- **ksp_gmres_restart** <restart>
- **pc_asm_overlap** <overlap>
- **pc_asm_type** [basic,restrict,interpolate,none]
## ACTS Numerical Tools: Functionality

<table>
<thead>
<tr>
<th>Computational Problem</th>
<th>Methodology</th>
<th>Algorithms</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Equations Problems</td>
<td>Direct</td>
<td>(LU) factorization</td>
<td>ScaLAPACK (dense)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cholesky factorization</td>
<td>SuperLU (sparse)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(LDLT) factorization (tridiagonal matrices)</td>
<td>ScaLAPACK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(QR) factorization</td>
<td>ScaLAPACK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(QR) with Column Pivoting factorization</td>
<td>ScaLAPACK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(LQ) factorization</td>
<td>ScaLAPACK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complete Orthogonal factorization</td>
<td>ScaLAPACK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Generalized (QR) factorization</td>
<td>ScaLAPACK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\bullet\ General \ QR, LQ, QL, RQ \ and RZ)</td>
<td>ScaLAPACK</td>
</tr>
</tbody>
</table>
## ACTS Numerical Tools: Functionality

<table>
<thead>
<tr>
<th>Computational Problem</th>
<th>Methodology</th>
<th>Algorithms</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Equations</td>
<td>Iterative</td>
<td>Conjugate Gradient (CG)</td>
<td>AztecOO (Trilinos)</td>
</tr>
<tr>
<td>Problems</td>
<td></td>
<td>GMRES</td>
<td>PETSc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CGS (CG Squared)</td>
<td>AztecOO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bi-CG-Stab</td>
<td>PETSc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Quasi-Minimal Residual (QMR)</td>
<td>AztecOO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Transpose Free QMR</td>
<td>AztecOO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SYMMLQ (symmetric LQ)</td>
<td>PETSc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Preconditioned CG</td>
<td>AztecOO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Richardson</td>
<td>PETSc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Block Jacobi preconditioner</td>
<td>AztecOO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Point Jacobi preconditioner</td>
<td>AztecOO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Least-squares polynomials</td>
<td>AztecOO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SOR preconditioner</td>
<td>PETSc</td>
</tr>
</tbody>
</table>
### ACTS Numerical Tools: Functionality

<table>
<thead>
<tr>
<th>Computational Problem</th>
<th>Methodology</th>
<th>Algorithms</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Equations Problems</td>
<td>iterative (cont.)</td>
<td>Overlapping Additive Schwarz (ASM) preconditioner</td>
<td>PETSc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Approximate Inverse</td>
<td>Hypre</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sparse LU preconditioner</td>
<td>AztecOO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Incomplete LU (ILU) preconditioner</td>
<td>AztecOO</td>
</tr>
<tr>
<td></td>
<td>Multigrid (MG)</td>
<td>MG preconditioner</td>
<td>Hypre PETSc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Algebraic Multigrid</td>
<td>ML (Trilinos)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Semicoarsening</td>
<td>Hypre</td>
</tr>
</tbody>
</table>
Structure of PETSc

- PETSc PDE Application Codes
  - ODE Integrators
  - Visualization Interface
  - Nonlinear Solvers, Unconstrained Minimization
  - Linear Solvers Preconditioners + Krylov Methods
  - Object-Oriented Matrices, Vectors, Indices
  - Grid Management
- Profiling Interface
  - Computation and Communication Kernels
    - MPI, MPI-IO, BLAS, LAPACK
### PETSc Numerical Components

#### Nonlinear Solvers

<table>
<thead>
<tr>
<th>Newton-based Methods</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line Search</td>
<td>Trust Region</td>
</tr>
</tbody>
</table>

#### Time Steppers

<table>
<thead>
<tr>
<th>Euler</th>
<th>Backward Euler</th>
<th>Pseudo Time Stepping</th>
<th>Other</th>
</tr>
</thead>
</table>

#### Krylov Subspace Methods

<table>
<thead>
<tr>
<th>GMRES</th>
<th>CG</th>
<th>CGS</th>
<th>Bi-CG-STAB</th>
<th>TFQMR</th>
<th>Richardson</th>
<th>Chebychev</th>
<th>Other</th>
</tr>
</thead>
</table>

#### Preconditioners

<table>
<thead>
<tr>
<th>Additive Schwartz</th>
<th>Block Jacobi</th>
<th>Jacobi</th>
<th>ILU</th>
<th>ICC</th>
<th>LU (Sequential only)</th>
<th>Others</th>
</tr>
</thead>
</table>

#### Matrices

<table>
<thead>
<tr>
<th>Compressed Sparse Row (AIJ)</th>
<th>Blocked Compressed Sparse Row (BAIJ)</th>
<th>Block Diagonal (BDIAG)</th>
<th>Dense</th>
<th>Matrix-free</th>
<th>Other</th>
</tr>
</thead>
</table>

#### Index Sets

<table>
<thead>
<tr>
<th>Indices</th>
<th>Block Indices</th>
<th>Stride</th>
<th>Other</th>
</tr>
</thead>
</table>

#### Vectors

#### Distributed Arrays
The DOE ACTS Collection (http://acts.nersc.gov)

CS267 - University of California, Berkeley
Partial list of Linear Solvers in PETSc

**Krylov Methods (KSP)**
- Conjugate Gradient
- GMRES
- CG-Squared
- Bi-CG-stab
- Transpose-free QMR
- etc.

**Preconditioners (PC)**
- Block Jacobi
- Overlapping Additive Schwarz
- ICC, ILU via BlockSolve95
- ILU(k), LU (sequential only)
- etc.
PETSc Example - Basic Linear Solver in C

```c
SLES  sles;               /* linear solver context */
Mat   A;                   /* matrix */
Vec   x, b;                /* solution, RHS vectors */
int   n, its;              /* problem dimension, number of iterations */

MatCreate(MPI_COMM_WORLD,PETSC_DECIDE,PETSC_DECIDE,
          n,n,&A);                 /* assemble matrix */
VecCreate(MPI_COMM_WORLD,PETSC_DECIDE,n,&x);
VecDuplicate(x,&b);         /* assemble RHS vector */

SLESCreate(MPI_COMM_WORLD,&sles);
SLESSetOperators(sles,A,A,DIFFERENT_NONZERO_PATTERN);
SLESSetFromOptions(sles);
SLESSolve(sles,b,x,&its);
SLESDestroy(sles);
```

Original Matrix from Linear System
Preconditioning matrix

SAME_NON_ZERO_PATTERN
SAME_PRECONDITIONER
Hypre Conceptual Interfaces

Linear System Interfaces

Linear Solvers

Data Layout

structured  composite  block-struc  unstruc  CSR

GMG, ...  FAC, ...  Hybrid, ...  AMGe, ...  ILU, ...
Hypre Conceptual Interfaces

- Structured-Grid Interface (Struct)
  - applications with logically rectangular grids

- Semi-Structured-Grid Interface (SSstruct)
  - applications with grids that are mostly—but not entirely—structured (e.g., block-structured, structured AMR, overset)

- Finite Element Interface (FEI)
  - unstructured-grid, finite element applications

- Linear-Algebraic Interface (IJ)
  - applications with sparse linear systems
Hypre Conceptual Interfaces

• Before writing your code:
  • choose a conceptual interface
  • choose a solver / preconditioner
  • choose a matrix type that is compatible with your solver / preconditioner and conceptual interface

• Code Development
  • build auxiliary structures (e.g., grids, stencils)
  • build matrix/vector through conceptual interface
  • build solver/preconditioner
  • solve the system
  • get desired information from the solver
Hypre’s IJ interface: setting up the Solver

List of Solvers and Preconditioners per Conceptual Interface

<table>
<thead>
<tr>
<th>Solvers</th>
<th>System Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Struct</td>
</tr>
<tr>
<td>Jacobi</td>
<td>X</td>
</tr>
<tr>
<td>SMG</td>
<td>X</td>
</tr>
<tr>
<td>PFMG</td>
<td>X</td>
</tr>
<tr>
<td>BoomerAMG</td>
<td>X</td>
</tr>
<tr>
<td>ParaSails</td>
<td>X</td>
</tr>
<tr>
<td>PILUT</td>
<td>X</td>
</tr>
<tr>
<td>Euclid</td>
<td>X</td>
</tr>
<tr>
<td>PCG</td>
<td>X</td>
</tr>
<tr>
<td>GMRES</td>
<td>X</td>
</tr>
</tbody>
</table>
### ACTS Numerical Tools: Functionality

<table>
<thead>
<tr>
<th>Computational Problem</th>
<th>Methodology</th>
<th>Algorithms</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Linear Least Squares Problems</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Least squares solution</td>
<td>minimize$_{x}$$|b - Ax|_2$</td>
<td>ScaLAPACK</td>
<td></td>
</tr>
<tr>
<td>Minimum norm solution</td>
<td>minimize$_{x}$$|x|_2$</td>
<td>ScaLAPACK</td>
<td></td>
</tr>
<tr>
<td>Minimum norm least squares</td>
<td>minimize$_{x}$$|x|_2$ and $|b - Ax|_2$</td>
<td>ScaLAPACK</td>
<td></td>
</tr>
<tr>
<td><strong>Standard Eigenvalue Problem</strong></td>
<td>Symmetric Eigenvalue Problems</td>
<td>$A_{z} = \lambda_{z}$ for $A = A^T$ or $A = A^H$</td>
<td>ScaLAPACK (dense)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SLEPc (sparse)</td>
</tr>
<tr>
<td><strong>Singular Value Problems</strong></td>
<td>Singular Value Decomposition</td>
<td>$A = UDV^T$</td>
<td>ScaLAPACK (dense)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$A = UDV^H$</td>
<td>SLEPc (sparse)</td>
</tr>
<tr>
<td><strong>Generalized Symmetric Definite Eigenproblem</strong></td>
<td>Eigenproblem</td>
<td>$A_{z} = \lambda B_{z}$, $AB_{z} = \lambda z$, $BA_{z} = \lambda z$</td>
<td>ScaLAPACK (dense)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SLEPc (sparse)</td>
</tr>
<tr>
<td><strong>Non-linear Equations Problems</strong></td>
<td>Newton-based</td>
<td>Line Search</td>
<td>PETSc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Trust Regions</td>
<td>PETSc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pseudo-transient continuation</td>
<td>PETSc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Matrix free</td>
<td>PETSc</td>
</tr>
<tr>
<td><strong>Non-linear Optimization Problems</strong></td>
<td>Newton-based</td>
<td>Newton</td>
<td>OPT++</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TAO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Finite Difference Newton</td>
<td>OPT++</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Quasi Newton</td>
<td>TAO (LMVM)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Nonlinear Interior Point</td>
<td>OPT++</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TAO</td>
</tr>
</tbody>
</table>
PETSc Nonlinear Solver (SNES)

Main Routine

Nonlinear Solvers (SNES)

Linear Solvers (SLES)

Solve \( F(u) = 0 \)

Application Initialization

Function Evaluation

Jacobian Evaluation

Post-Processing

PETSc

User code

PETSc code
Time Dependent PDE Solution

Main Routine

Timestepping Solvers (TS)

Nonlinear Solvers (SNES)

Linear Solvers (SLES)

Application Initialization

Function Evaluation

Jacobian Evaluation

Post-Processing

PETSc

Solve

\[ U_t = F(U, U_x, U_{xx}) \]
# ACTS Numerical Tools: Functionality

<table>
<thead>
<tr>
<th>Computational Problem</th>
<th>Methodology</th>
<th>Algorithms</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-linear Optimization Problems (cont.)</td>
<td>CG</td>
<td>Standard non-linear CG</td>
<td>OPT++</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Limited memory BFGS</td>
<td>TAO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gradient Projection</td>
<td></td>
</tr>
<tr>
<td>Ordinary differential equations</td>
<td>Direct Search</td>
<td>Without derivative Information</td>
<td>OPT++</td>
</tr>
<tr>
<td></td>
<td>Semismooth</td>
<td>Infeasible semismooth</td>
<td>TAO</td>
</tr>
<tr>
<td></td>
<td>Integration</td>
<td>Feasible semismooth</td>
<td>TAO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Variable coefficient forms of the Adams-Moulton</td>
<td>CVODE (SUNDIALS)</td>
</tr>
<tr>
<td></td>
<td>Backward differential formula</td>
<td>Direct and iterative solvers</td>
<td>CVODE (SUNDIALS)</td>
</tr>
<tr>
<td>Nonlinear Algebraic Equations</td>
<td>Inexact Newton</td>
<td>Line search</td>
<td>KINSOL (SUNDIALS)</td>
</tr>
<tr>
<td>Differential-Algebraic Equations</td>
<td>Backward differential formula</td>
<td>Direct and iterative solvers</td>
<td>IDA (SUNDIALS)</td>
</tr>
</tbody>
</table>
Classes of Problems in OPT++

- Four major classes of problems available
  - \(NLF0(\text{ndim}, \text{fcn}, \text{init}_\text{fcn}, \text{constraint})\)
    - Basic nonlinear function, no derivative information available
  - \(NLF1(\text{ndim}, \text{fcn}, \text{init}_\text{fcn}, \text{constraint})\)
    - Nonlinear function, first derivative information available
  - \(FDNLF1(\text{ndim}, \text{fcn}, \text{init}_\text{fcn}, \text{constraint})\)
    - Nonlinear function, first derivative information approximated
  - \(NLF2(\text{ndim}, \text{fcn}, \text{init}_\text{fcn}, \text{constraint})\)
    - Nonlinear function, first and second derivative information available
Classes of Solvers in OPT++

- Direct search
  - No derivative information required
- Conjugate Gradient
  - Derivative information may be available but doesn’t use quadratic information
- Newton-type methods
  - Algorithm attempts to use/approximate quadratic information
  - Newton
  - Finite-Difference Newton
  - Quasi-Newton
  - NIPS
TAO - Interface with PETSc
TAO - Bound Constraint Optimization

• Conjugate Gradient
• Limited-Memory variable-metric algorithms
• Newton Algorithms where user supplies

The user must provide:

• The Function and its First Derivative
• For Newton Methods the second Derivative is also required (Hessian Matrix)
\[
x_{k+1} = x_k + \alpha_k p_k
\]
\[
p_{k+1} = -\nabla f(x_k) + \beta_k p_k
\]
where \(\alpha_k\) is determined by a line search.

Three choices of \(\beta_k\) are possible \((g_k = \nabla f(x_k)):\)

\[
\beta_{k}^{FR} = \left( \frac{\|g_{k+1}\|}{\|g_k\|} \right)^2, \quad \text{Fletcher-Reeves}
\]
\[
\beta_{k}^{PR} = \frac{\langle g_{k+1}, g_{k+1} - g_k \rangle}{\|g_k\|^2}, \quad \text{Polak-Rivière}
\]
\[
\beta_{k}^{PR+} = \max \{\beta_{k}^{PR}, 0\}, \quad \text{PR-plus}
\]
\[ x_{k+1} = x_k - \alpha_k H_k \nabla f(x_k) \]

where \( \alpha_k \) is determined by a line search.

The matrix \( H_k \) is defined in terms of information gathered during the previous \( m \) iterations.

- \( H_k \) is positive definite.
- Storage of \( H_k \) requires \( 2mn \) locations.
- Computation of \( H_k \nabla f(x_k) \) costs \( (8m + 1)n \) flops.
At each iteration the step $s_k$ (approximately) minimizes

$$\min \{ q_k(x_k + s) : s_i = 0, \ i \in A_k, \ x_l \leq x_k + s \leq x_u, \ ||s|| \leq \Delta_k \}$$

where $q_k$ is the quadratic approximation,

$$q_k(w) = \langle \nabla f(x_k), w \rangle + \frac{1}{2} \langle w, \nabla^2 f(x_k)w \rangle,$$

to the function, and $\Delta_k$ is the trust region bound.

- Predict an active set $A_k$.
- Compute a step $s_k$
- $x_{k+1} = x_k + s_k$ if $f(x_k + s_k) < f(x_k)$, otherwise $x_{k+1} = x_k$.
- Update $\Delta_k$. 
What codes are being developed?

- **Global Arrays**
  - Parallel programs that use large distributed arrays
  - Support for Grids and meshes
  - Language Interoperability

- **Overture**

- **PAWS**
  - Coupling distributed applications
  - On-line visualization and computational steering

- **Chasm**

- **CUMULVS**

- **Globus**

- **TAU**

- **Globus**

- **Overture**

- **PAWS**

- **Chasm**

- **CUMULVS**

- **Globus**

- **TAU**
Tool Interoperability
Tool-to-Tool

PETSc

Ex 1

TOOL A

TOOL B

TOOL C

TOOL D

TOOL E

TOOL F

TAU

Ex 2
Component Technology!

Tool A

Tool B

Tool C

Tool D

CCA

ESI
PSE's and Frameworks

User

View_field(T1)

Ax = b

Az = λz

A = UΣVT

High Level Interfaces

PMatlab
PyACTS

OPT++ PAWS Globus CUMULVS TAU
AZTEC Hypre PETSc Chombo Global Arrays
ScaLAPACK SuperLU TAO PVODE Overture

Ax = b

A = UΣVT
```python
% run 4

submit: Processed command file through Submit Filter: "/usr/common/nsg/etc/subfilter".

>>> import sys
>>> sys.path.append('/u6/nkang/kn/pyacts_1/build/lib.aix-5.1-mpi-2.2')
>>> import scalapack
>>> scalapack.ex2("ex2_mat","ex2_rhs","sol",6,1,2,2,2,1)

Scalapack Example Program #2 (C-version) -- 07/24/2003
Solving AX=B
where A is a 6 by 6 matrix,
B is a 6 by 1 matrix,
with a block size of 2
Running on 4 processes, where the process grid is 2 by 2
INFO code returned by PDGESV = 0

According to the normalized residual the solution is correct.

\[ \frac{||AX-B||}{||X||} \leq \text{eps} \times N \]
\[ = 1.25878215e-01 \]

The solution is written to file sol

End of test.
```
More Tutorials

http://acts.nersc.gov/events/Workshop2003/

SIAM PP04 Short Course + Workshop 2003 CD
Request: acts-support@nersc.gov - It’s FREE!

Coming Soon! ACTS Workshop 2004