The Petra Object Model: Useful Parallel Abstractions

Michael A. Heroux
Sandia National Laboratories

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company,
for the United States Department of Energy under contract DE-AC04-94AL85000.

Alternate Title

"In absence of a portable parallel language and in the presence of recurring yet evolving system architectures, while in the meantime trying to get work done with the current tools and solve problems of interest, what we have developed to solve large-scale systems of equations on large-scale computers and allow for adapting in the future."

Sandia National Laboratories

Outline

- Context for Discussion: Trilinos and POM.
- Parallel Basic Linear Algebra Software Requirements.
- Main Points:
  - Abstract machine model provides flexibility.
  - Abstraction of distribution is essential.
  - Abstraction of high-level operators enable use of wide range of operators.
  - (In the solvers area) Nested Parallelism is a Dubious Option.

Sandia National Laboratories

Trilinos Contributors

Chris Baker
Developer of Anasazi, RBGen

Ross Bartlett
Lead Developer of MOOCHO, Stratimikos, RTOp, Thyra
Developer of Rythmos

Paul Boggs
Developer of Thyra

Erik Boman
Lead Developer Isorropia

Todd Coffey
Lead Developer of Rythmos

Jason Cross
Developer of Jpetra

David Day
Developer of Komplex

Clark Dohrmann
Lead developer of CLAPS

Michael Gee
Developer of ML, Moertel, NOX

Bob Heaphy
Lead developer of Trilinos SQA

Mike Heroux
Trilinos Project Leader
Lead Developer of Epetra, AztecOO, Kokkos, Komplex, IFPACK, Thyra, Tpetra
Developer of Amesos, Belos, EpetraExt, Jpetra

Ulrich Hetmaniuk
Developer of Anasazi

Robert Hoekstra
Lead Developer of EpetraExt
Developer of Epetra, Thyra, Tpetra

Russell Hooper
Developer of NOX

Vicki Howle
Lead Developer of Meros
Developer of Belos and Thyra

Jonathan Hu
Developer of ML

Sarah Knepper
Developer of Komplex

Tammy Kolda
Lead Developer of NOX

Joe Kotulski
Lead Developer of Pliris

Rich Lehoucq
Developer of Anasazi and Belos

Kevin Long
Lead Developer of Thyra
Developer of Belos and Teuchos

Roger Pawlowski
Lead Developer of NOX

Michael Phenow
Trilinos Webmaster
Lead Developer of New_Package
Developer WebTrilinos

Eric Phipps
Lead developer Sacado
Developer of LOCA, NOX

Dennis Ridzal
Lead Developer of Aristos

Marzio Sala
Lead Developer of Didasko, Galeri, IFPACK, WebTrilinos
Developer of ML, Amesos

Andrew Salinger
Lead Developer of LOCA, Capo

Paul Sexton
Developer of Epetra and Tpetra

Bob Shuttleworth
Developer of Meros.

Chris Siefert
Developer of ML

Bill Spotz
Lead Developer of PyTrilinos
Developer of Epetra, New_Package

Ken Stanley
Lead Developer of Amesos and New_Package

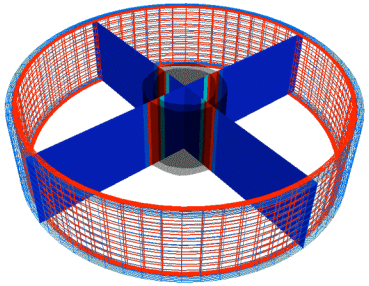
Heidi Thornquist
Lead Developer of Anasazi, Belos, RBGen and Teuchos

Ray Tuminaro
Lead Developer of ML and Meros

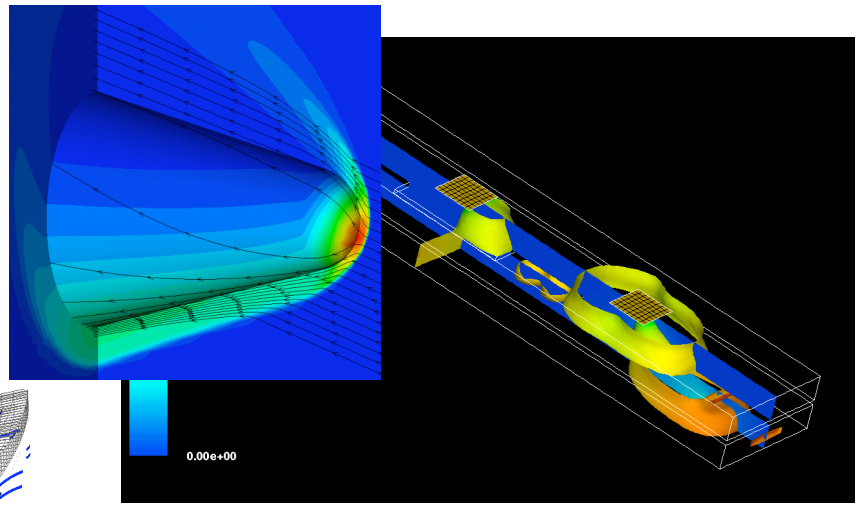
Jim Willenbring
Developer of Epetra and New_Package.
Trilinos library manager

Alan Williams
Lead Developer Isorropia
Developer of Epetra, EpetraExt, AztecOO, Tpetra

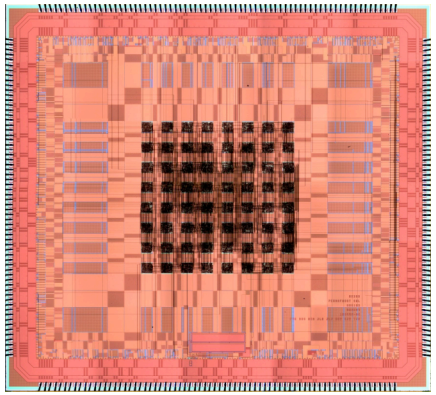
Target Problems: PDES and more…

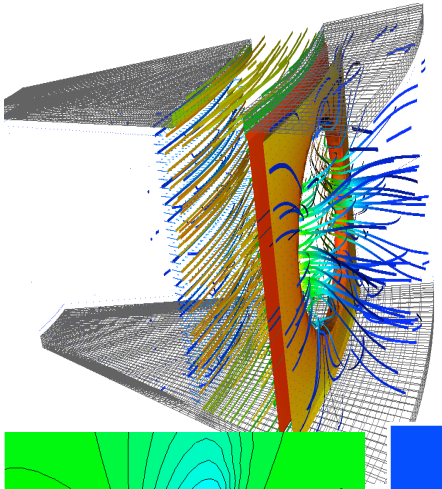


PDES

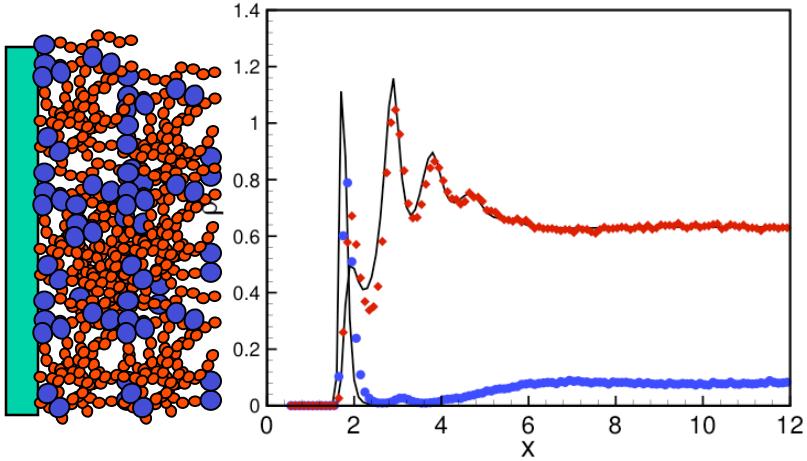


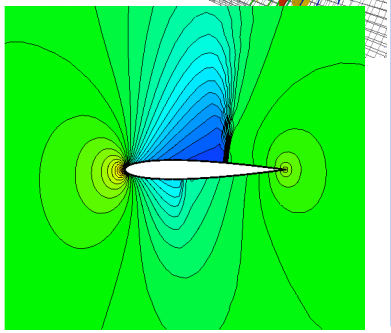
Circuits

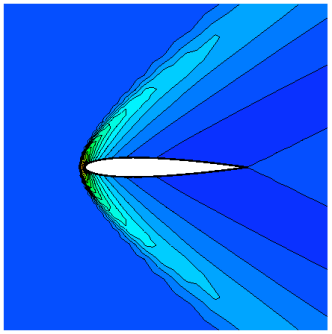




Inhomogeneous Fluids







And More…

Trilinos Packages

- Trilinos is a collection of *Packages*.
- Each package is:
  - Focused on important, state-of-the-art algorithms in its problem regime.
  - Developed by a small team of domain experts.
  - Self-contained: No explicit dependencies on any other software packages (with some special exceptions).
  - Configurable/buildable/documented on its own.
- Sample packages: NOX, AztecOO, ML, IFPACK, Meros.
- Special package collections:
  - Petra (Epetra, Tpetra, Jpetra): Concrete Data Objects
  - Thyra: Abstract Conceptual Interfaces
  - Teuchos: Common Tools.
  - New_package: Jumpstart prototype.

Sandia National Laboratories

Trilinos Package Summary

Objective	Package(s)
Linear algebra objects	Epetra, Jpetra, Tpetra
Krylov solvers	AztecOO, Belos, Komplex
ILU-type preconditioners	AztecOO, IFPACK
Multilevel preconditioners	ML, CLAPS
Eigenvalue problems	Anasazi
Block preconditioners	Meros
Direct sparse linear solvers	Amesos
Direct dense solvers	Epetra, Teuchos, Pliris
Abstract interfaces	Thyra
Nonlinear system solvers	NOX, LOCA
Time Integrators/DAEs	**Rythmos**
C++ utilities, (some) I/O	Teuchos, EpetraExt, Kokkos
Trilinos Tutorial	Didasko
"Skins"	PyTrilinos, **WebTrilinos**, Star-P, **Stratimikos, *ForTrilinos***
Optimization	**MOOCHO, Aristos**
Archetype package	NewPackage
Other new in 7.0 *(8.0)*	**Galeri, Isorropia, Moertel, RTOp, *Aristos, RBGen***

Full "Vertical" Solver Coverage

Trilinos Packages

· Optimization Problems: · Unconstrained: · Constrained:	 Find $u \in \Re^n$ that minimizes $f(u)$ Find $y \in \Re^m$ and $u \in \Re^n$ that minimizes $f(y,u)$ s.t. $c(y,u) = 0$	MOOCHO
· Transient Problems: · DAEs/ODEs:	Solve $f(\dot{x}(t), x(t), t) = 0$ $t \in [0,T],\ x(0) = x_0,\ \dot{x}(0) = x_0'$ for $x(t) \in \Re^n,\ t \in [0,T]$	Rythmos
· Nonlinear Problems: · Nonlinear equations: · Stability analysis:	Given nonlinear op $c(x,u) \in \Re^{n+m} \rightarrow \Re^n$ Solve $c(x) = 0$ for $x \in \Re^n$ For $c(x,u) = 0$ find space $u \in U \ni \frac{\partial c}{\partial x}$ singular	NOX LOCA
· Implicit Linear Problems: · Linear equations: · Eigen problems:	Given linear ops (matrices) $A, B \in \Re^{n \times n}$ Solve $Ax = b$ for $x \in \Re^n$ Solve $Av = \lambda Bv$ for (all) $v \in \Re^n,\ \lambda \in \Re$	AztecOO, Belos, Ifpack,ML,etc. Anasazi
· Explicit Linear Problems: · Matrix/graph equations: · Vector problems:	 Compute $y = Ax;\ A = A(G);\ A \in \Re^{m \times n}, G \in \square^{m \times n}$ Compute $y = \alpha x + \beta w;\ \alpha =< x, y >;\ x, y \in \Re^n$	Epetra, Tpetra

Observations

- 30-80% of application runtime spent in solvers or:
- Solvers are not applications but can drive app performance.

- Vast majority of solver time spent in kernels or:
- Kernels are not solvers but can drive solver performance.

Sandia National Laboratories

A Simple Epetra/AztecOO Program

```
// Header files omitted…
int main(int argc, char *argv[]) {
Epetra_SerialComm Comm();
```

```
// ***** Map puts same number of equations on each pe *****

  int NumMyElements = 1000 ;
  Epetra_Map Map(-1, NumMyElements, 0, Comm);
  int NumGlobalElements = Map.NumGlobalElements();
```

```
// ***** Create an Epetra_Matrix  tridiag(-1,2,-1) *****

  Epetra_CrsMatrix A(Copy, Map, 3);
  double negOne = -1.0; double posTwo = 2.0;

  for (int i=0; i<NumMyElements; i++) {
    int GlobalRow = A.GRID(i);
    int RowLess1 = GlobalRow - 1;
    int RowPlus1 = GlobalRow + 1;
    if (RowLess1!=-1)
      A.InsertGlobalValues(GlobalRow, 1, &negOne, &RowLess1);
    if (RowPlus1!=NumGlobalElements)
      A.InsertGlobalValues(GlobalRow, 1, &negOne, &RowPlus1);
    A.InsertGlobalValues(GlobalRow, 1, &posTwo, &GlobalRow);
  }
A.FillComplete(); // Transform from GIDs to LIDs
```

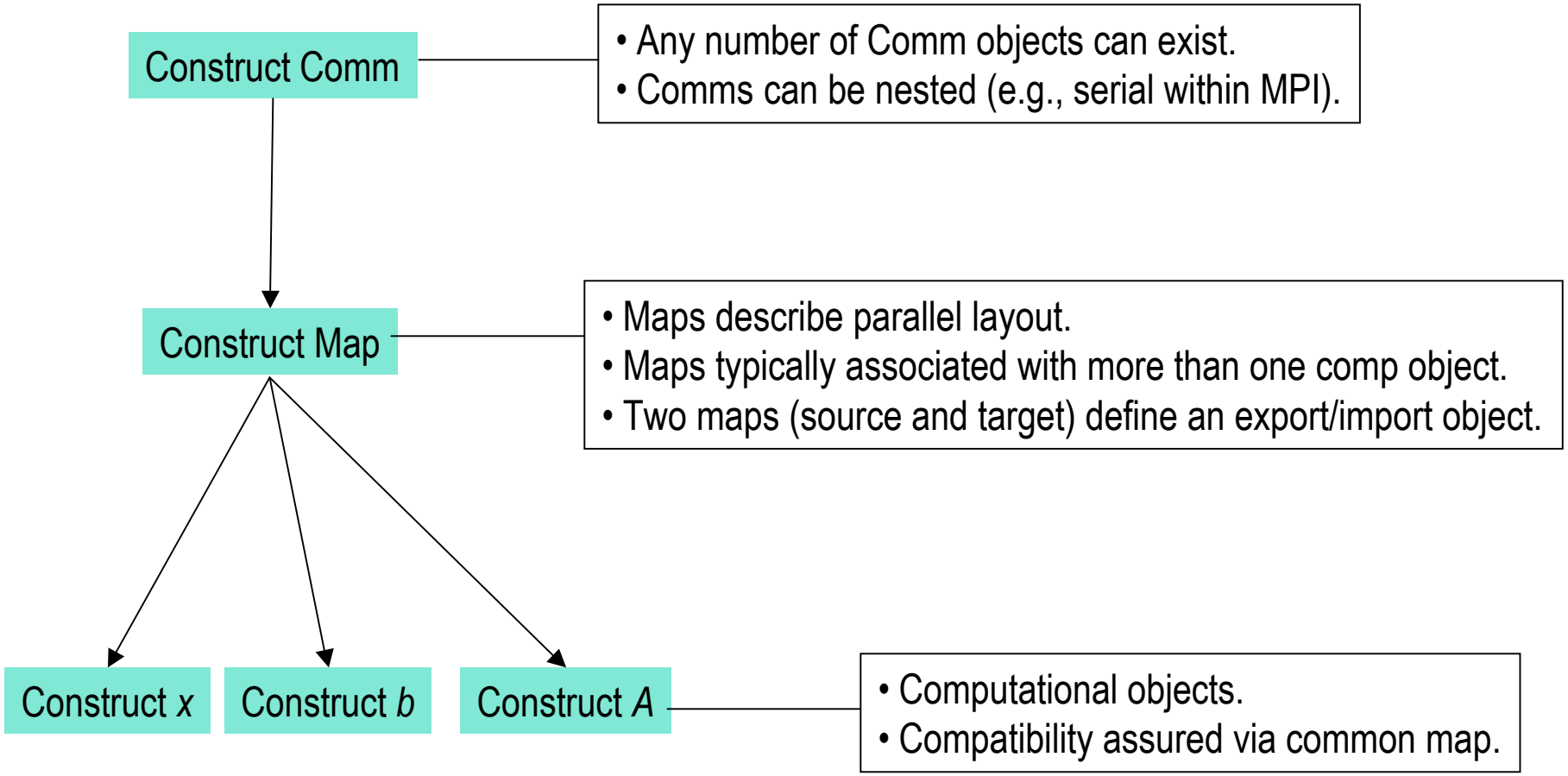
```
  // ***** Create x and b vectors *****
  Epetra_Vector x(Map);
  Epetra_Vector b(Map);
  b.Random(); // Fill RHS with random #s
```

```
// ***** Create Linear Problem *****
  Epetra_LinearProblem problem(&A, &x, &b);
```

```
  // ***** Create/define AztecOO instance, solve *****
  AztecOO solver(problem);
  solver.SetAztecOption(AZ_precond, AZ_Jacobi);
  solver.Iterate(1000, 1.0E-8);
```

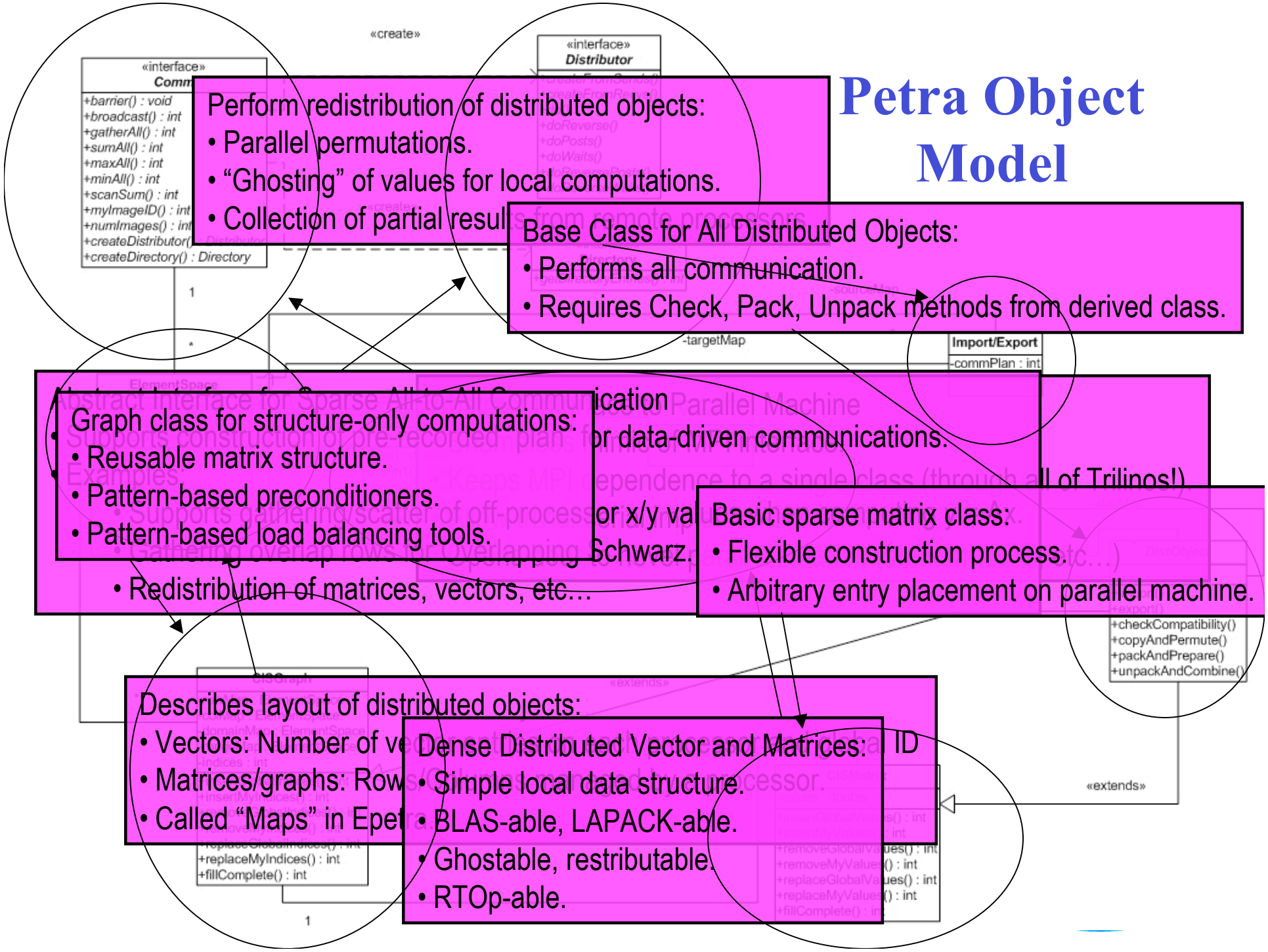
```
// ***** Report results, finish **********************
  cout << "Solver performed " << solver.NumIters()
       << " iterations." << endl
       << "Norm of true residual = "
       << solver.TrueResidual()
       << endl;


  return 0;
}
```

Typical Flow of Epetra Object Construction



Sandia National Laboratories

Petra Object Model



Perform redistribution of distributed objects:
- Parallel permutations.
- "Ghosting" of values for local computations.
- Collection of partial results from remote processors.

Base Class for All Distributed Objects:
- Performs all communication.
- Requires Check, Pack, Unpack methods from derived class.

Abstract Interface for Sparse All-to-All Communication
- Supports construction of pre-recorded "plan" for data-driven communications.
- Examples:
  - Supports gathering/scatter of off-processor x/y values for Ax.
  - Gathering overlap rows for Overlapping Schwarz.
  - Redistribution of matrices, vectors, etc…

Graph class for structure-only computations:
- Reusable matrix structure.
- Pattern-based preconditioners.
- Pattern-based load balancing tools.

Basic sparse matrix class:
- Flexible construction process.
- Arbitrary entry placement on parallel machine.

Describes layout of distributed objects:
- Vectors: Number of vectors and vector and processor ID
- Matrices/graphs: Rows/Columns assigned by processor.
- Called "Maps" in Epetra.

Dense Distributed Vector and Matrices:
- Simple local data structure.
- BLAS-able, LAPACK-able.
- Ghostable, restributable.
- RTOp-able.

Petra Implementations

- Three version under development:
- **Epetra** (Essential Petra):
  - Current production version.
  - Restricted to real, double precision arithmetic.
  - Uses stable core subset of C++ (circa 2000).
  - Interfaces accessible to C and Fortran users.
- **Tpetra** (Templated Petra):
  - Next generation C++ version.
  - Templated scalar and ordinal fields.
  - Uses namespaces, and STL: Improved usability/efficiency.
- **Jpetra** (Java Petra):
  - Pure Java. Portable to any JVM.
  - Interfaces to Java versions of MPI, LAPACK and BLAS via interfaces.

Sandia National Laboratories

Important Requirements for Parallel Basic Linear Algebra Software

- Major algorithmic requirements include:
  - [A1] Support sparse and dense matrix construction, use and reuse in scientific and engineering applications.
  - [A2] Support for advanced numerical algorithms.
  - [A3] Abstractions of basic linear algebra objects for efficiency and extensibility.
- **Major software design requirements include:**
  - **[S1] Efficient execution on parallel computers.**
  - **[S2] Portability of libraries on current and future serial and parallel computers.**
  - **[S3] Utilize existing software capabilities for performance and functionality.**
- Two usability requirements, that are not entirely compatible, are:
  - [U1] Accessible to C and Fortran applications and users.
  - [U2] Efficient support of multiple data types including 32 or 64-bit floating point, complex numbers and large integer types.

Sandia National Laboratories

First Useful Abstraction: Comm

Sandia National Laboratories

Epetra Communication Classes

- Epetra_Comm is a pure virtual class:
  - Has no executable code: Interfaces only.
  - Encapsulates behavior and attributes of the parallel machine.
  - Defines interfaces for basic services such as:
    - Collective communications.
    - Gather/scatter capabilities.
  - Allows multiple parallel machine implementations.
- Implementation details of parallel machine confined to Comm classes.
- In particular, rest of Epetra (and rest of Trilinos) has no dependence on MPI.

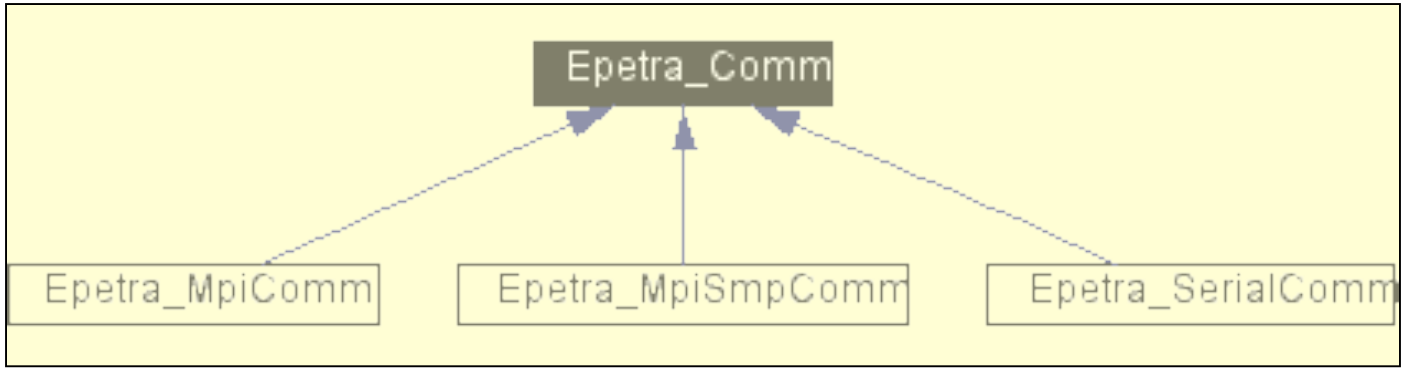
Sandia National Laboratories

Comm Methods

- CreateDistributor() const=0 [pure virtual]
- CreateDirectory(const Epetra_BlockMap & map) const=0 [pure virtual]
- Barrier() const=0 [pure virtual]
- Broadcast(double *MyVals, int Count, int Root) const=0 [pure virtual]
- Broadcast(int *MyVals, int Count, int Root) const=0 [pure virtual]
- GatherAll(double *MyVals, double *AllVals, int Count) const=0 [pure virtual]
- GatherAll(int *MyVals, int *AllVals, int Count) const=0 [pure virtual]
- MaxAll(double *PartialMaxs, double *GlobalMaxs, int Count) const=0 [pure virtual]
- MaxAll(int *PartialMaxs, int *GlobalMaxs, int Count) const=0 [pure virtual]
- MinAll(double *PartialMins, double *GlobalMins, int Count) const=0 [pure virtual]
- MinAll(int *PartialMins, int *GlobalMins, int Count) const=0 [pure virtual]
- MyPID() const=0 [pure virtual]
- NumProc() const=0 [pure virtual]
- Print(ostream &os) const=0 [pure virtual]
- ScanSum(double *MyVals, double *ScanSums, int Count) const=0 [pure virtual]
- ScanSum(int *MyVals, int *ScanSums, int Count) const=0 [pure virtual]
- SumAll(double *PartialSums, double *GlobalSums, int Count) const=0 [pure virtual]
- SumAll(int *PartialSums, int *GlobalSums, int Count) const=0 [pure virtual]
- ~Epetra_Comm() [inline, virtual]

Sandia National Laboratories

Comm Implementations



Three current implementations of Petra_Comm:

- Epetra_SerialComm:
  - Allows easy simultaneous support of serial and parallel version of user code.
- Epetra_MpiComm:
  - OO wrapping of C MPI interface.
- Epetra_MpiSmpComm:
  - Allows definition/use of shared memory multiprocessor nodes.

Sandia National Laboratories

Comm, Distributor, Directory

- Efficient Execution:
  - Provide high-level abstract description of parallel machine:
    - Flexibility: How the operations optimally performed.
    - Distributor: Static, reusable, record of data-dependent communications.
- Portability on Parallel Platforms:
  - All classes abstract.
  - Can provide multiple implementations: MPI and serial.
  - No widespread dependence on MPI.
  - **Possible to develop adapters for other parallel libraries and languages.**
- Utilize Existing Software:
  - Utilize MPI as the primary adapter
  - Provide trivial serial adapters.

Sandia National Laboratories

Example: Specialized Comm Adapters

```
int levelsetSolve_Epetra_Operator::Apply (const Epetra_MultiVector& X,
                                          Epetra_MultiVector& Y) const {
try {
const Epetra_SmpMpiComm & comm =
                    dynamic_cast<const Epetra_SmpMpiComm>(X.Map().Comm());
 … comm.getThread(…)
} catch (…)
```

- Fragment of code from levelset preconditioner Epetra_Operator adapter.
- Allows specialized parallel machine types.
- *Should* allow specializations such as:
  - CAF/UPC kernels
  - Co-processors: GPUs, Cell SPEs.

Sandia National Laboratories

Second Useful Abstraction: ElementSpace aka Map

Sandia National Laboratories

Map Classes

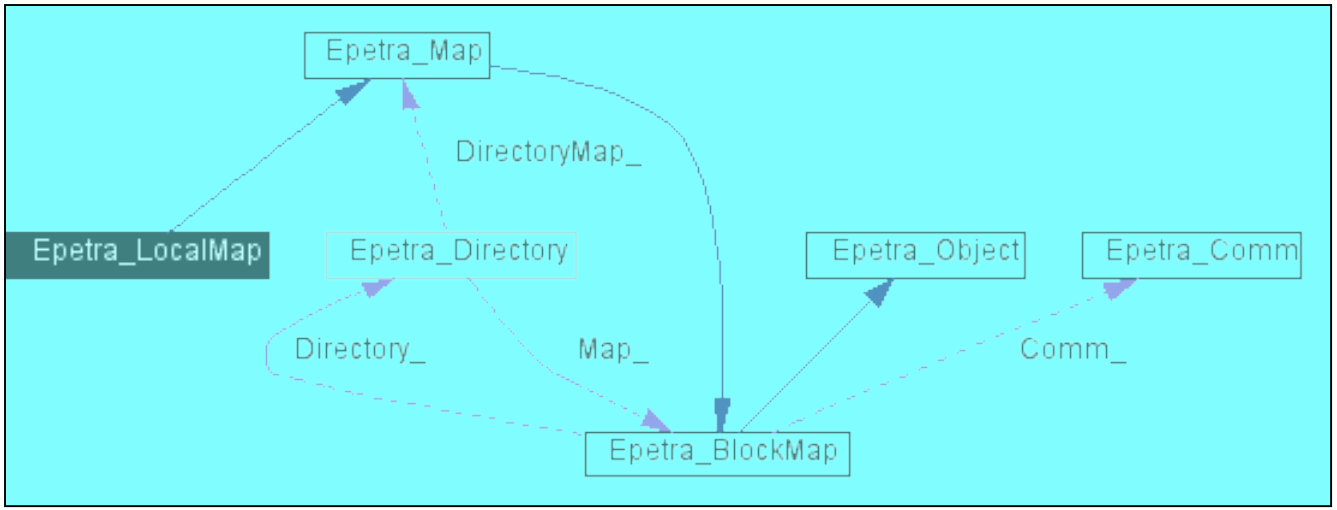
- Epetra maps prescribe the layout of distributed objects across the parallel machine.
- Typical map: 99 elements, 4 MPI processes could look like:
  - Number of elements = 25 on PE 0 through 2,
= 24 on PE 3.
  - GlobalElementList = {0, 1, 2, …, 24} on PE 0,
= {25, 26, …, 49} on PE 1. … etc.
- Funky Map: 10 elements, 3 MPI processes could look like:
  - Number of elements = 6 on PE 0,
= 4 on PE 1,
= 0 on PE 2.
  - GlobalElementList = {22, 3, 5, 2, 99, 54} on PE 0,
= { 5, 10, 12, 24} on PE 1,
= {} on PE 2.

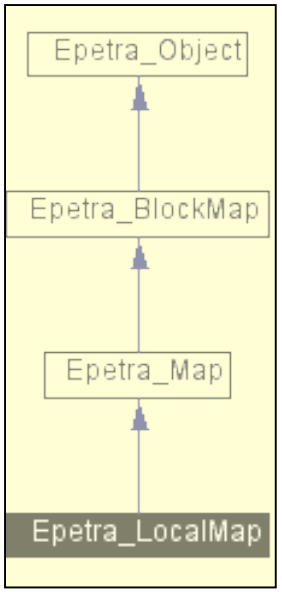
Note: Global elements IDs (GIDs) are only labels:

- Need not be contiguous range on a processor.
- Need not be uniquely assigned to processors.
- Funky map is not unreasonable, given auto-generated meshes, etc.
- Use of a "Directory" facilitates arbitrary GID support.

Sandia National Laboratories

Epetra Map Collaboration Diagram & Inheritance Graph





Notes:

1. **Epetra_Object is base class for all concrete Epetra classes:**
   - **Has labeling and ostream methods.**
   - **Maintains definitions of global constants.**
2. **BlockMap is the base map class.**
3. **Maps have Epetra_Directory to keep track of global ID distribution.**

Sandia National Laboratories

ElementSpace/Map

- Critical classes for efficient parallel execution. Provides ability to:
  - Generate families of compatible objects.
  - Quickly test if two objects have compatible layout.
  - Redistribute objects to make compatible layout (using Import/Export).

Sandia National Laboratories

Example: Quick Testing of Compatible Distributions

```
int dft_PolyA22_Epetra_Operator::Apply (const Epetra_MultiVector& X,
                                        Epetra_MultiVector& Y) const {

  TEST_FOR_EXCEPT(!X.Map().SameAs(OperatorDomainMap()));
  TEST_FOR_EXCEPT(!Y.Map().SameAs(OperatorRangeMap()));
```

- Fragment of code from Epetra_Operator adapter.
- Test for compatibility of maps.
- Lack of this ability is critical flaw of most parallel languages to date.

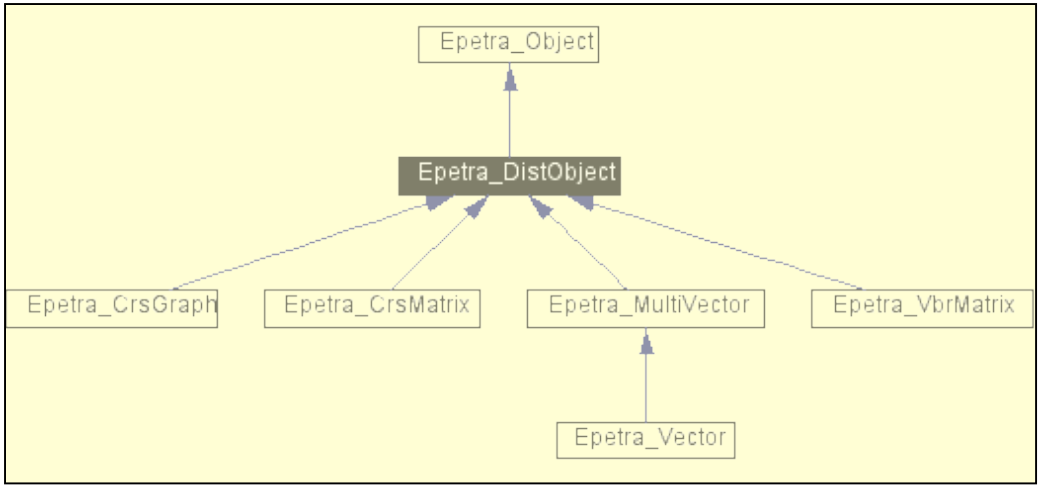
Sandia National Laboratories

Third Useful Abstraction: DistObject

Sandia National Laboratories

Epetra DistObject Base Class

- Some Epetra distributed object classes:
  - Vector
  - MultiVector
  - CrsGraph
  - CrsMatrix
  - VbrMatrix



- DistObject is a base class for all the above:
  - Construction of DistObject requires a Map (or BlockMap or LocalMap).
  - Has concrete methods for parallel data redistribution of an object.
  - Has virtual Pack/Unpack method that each derived class must implement.
- DistObject advantages:
  - Minimized redundant code.
  - Facilitates incorporation of other distributed objects in future.

Sandia National Laboratories

Epetra_DistObject Virtual Methods

virtual int CheckSizes (const Epetra_SrcDistObject &Source)=0

Allows the source and target (this) objects to be compared for compatibility, return nonzero if not.

virtual int CopyAndPermute (const Epetra_SrcDistObject &Source, int NumSameIDs,
int NumPermuteIDs, int *PermuteToLIDs, int *PermuteFromLIDs)=0

Perform ID copies and permutations that are on processor.

virtual int PackAndPrepare (const Epetra_SrcDistObject &Source, int NumExportIDs, int *ExportLIDs,
int Nsend, int Nrecv, int &LenExports, char *&Exports, int &LenImports,
char *&Imports, int &SizeOfPacket, Epetra_Distributor &Distor)=0

Perform any packing or preparation required for call to DoTransfer().

virtual int UnpackAndCombine (const Epetra_SrcDistObject &Source, int NumImportIDs, int *ImportLIDs,
char *Imports, int &SizeOfPacket, Epetra_Distributor &Distor,
Epetra_CombineMode CombineMode)=0

Perform any unpacking and combining after call to DoTransfer().

Sandia National Laboratories

Epetra_DistObject Import/Export Methods

int Import (const Epetra_SrcDistObject &A, const Epetra_Import &Importer, Epetra_CombineMode CombineMode)
Imports an Epetra_SrcDistObject using the Epetra_Import object.

int Import (const Epetra_SrcDistObject &A, const Epetra_Export &Exporter, Epetra_CombineMode CombineMode)
Imports an Epetra_SrcDistObject using the Epetra_Export object.

int Export (const Epetra_SrcDistObject &A, const Epetra_Import &Importer, Epetra_CombineMode CombineMode)
Exports an Epetra_SrcDistObject using the Epetra_Import object.

int Export (const Epetra_SrcDistObject &A, const Epetra_Export &Exporter, Epetra_CombineMode CombineMode)
Exports an Epetra_SrcDistObject using the Epetra_Export object.

Sandia National Laboratories

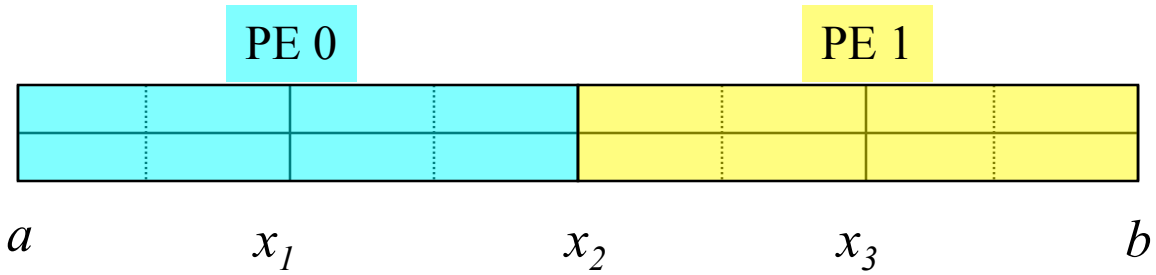
Import, Export and DistObject

- Work with any class that *isa* DistObject (or SrcDistObject).
- Abstraction of these operations to classes simplifies:
  - Optimal implementation on a variety of systems.
  - Reuse of complex parallel data redistribution:
    - DistObject has most of the complexity.

Sandia National Laboratories

Example: 1D Matrix Assembly

$$-u_{xx} = f$$
$$u(a) = \gamma_0$$
$$u(b) = \gamma_1$$



- 3 Equations: Find u at x_1, x_2 and x_3
- Equation for u at x_2 gets a contribution from PE 0 and PE 1.
- Would like to compute partial contributions independently.
- Then combine partial results.

Sandia National Laboratories

Two Maps

- We need two maps:
  - Assembly map:
    - PE 0: { 1, 2 }.
    - PE 1: { 2, 3 }.
  - Solver map:
    - PE 0: { 1, 2 } (we arbitrate ownership of 2).
    - PE 1: { 3 }.

Sandia National Laboratories

End of Assembly Phase

- At the end of assembly phase we have AssemblyMatrix:
  - On PE 0:

$$\begin{array}{l} \text{Equation 1:} \\ \text{Equation 2:} \end{array} \overset{\textit{AssemblyMatrix}}{\begin{bmatrix} 2 & -1 & 0 \\ -1 & 1 & 0 \end{bmatrix}} \quad \overset{\textit{AssemblyRhs}}{\begin{bmatrix} 1 \\ 1 \end{bmatrix}}$$

  - On PE 1:

$$\begin{array}{l} \text{Equation 2:} \\ \text{Equation 3:} \end{array} \begin{bmatrix} 0 & 1 & -1 \\ 0 & -1 & 2 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

- Want to assign all of Equation 2 to PE 0 for use with solver.

Sandia National Laboratories

Export Assembly Matrix to Solver Matrix

```
Epetra_Export Exporter(AssemblyMap, SolverMap);

Epetra_CrsMatrix SolverMatrix (Copy, SolverMap, 0);

SolverMatrix.Export(AssemblyMatrix, Exporter, Add);

SolverMatrix.TransformToLocal();
```

Sandia National Laboratories

End of Export Phase

- At the end of Export phase we have SolverMatrix:
  - On PE 0:

$$\begin{array}{lll} & \textit{SolverMatrix} & \textit{SolverRhs} \\ \text{Equation 1:} & \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \end{bmatrix} & \begin{bmatrix} 1 \\ 2 \end{bmatrix} \\ \text{On PE 1:} \quad \text{Equation 2:} & & \\ \text{Equation 3:} & \begin{bmatrix} 0 & -1 & 2 \end{bmatrix} & \begin{bmatrix} 1 \end{bmatrix} \end{array}$$

- Each row is uniquely owned.

Sandia National Laboratories

Need for Import/Export

- Solvers for complex engineering applications need expressive, easy-to-use parallel data redistribution:
  - Allows better scaling for non-uniform overlapping Schwarz.
  - Necessary for robust solution of multiphysics problems.
- We have found import and export facilities to be a very natural and powerful technique to address these issues.

Sandia National Laboratories

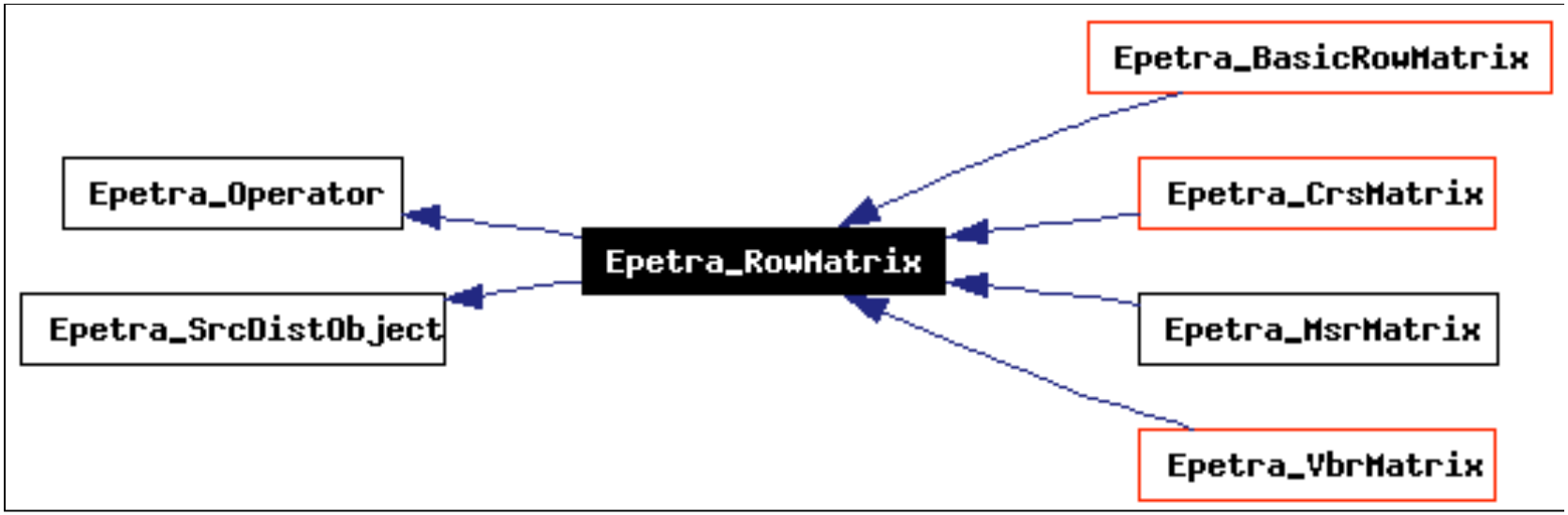
Other uses for Import/Export

- In addition, import and export facilities provide a variety of other capabilities:
  - Communication needs of sparse matrix multiplication.
  - Parallel assembly: Shared nodes receive contributions from multiple processors, reverse operation replicates results back.
  - Higher order interpolations are easy to implement.
  - Ghost node distributions.
  - Changing work loads can be re-balanced.
  - Sparse matrix transpose become trivial to implement.
  - Allows gradual MPI-izing of an application.
  - Cached Overlapped distributed vectors (generalization of distributed sparse MV).
  - Rendezvous algorithms easy to implement.

Sandia National Laboratories

Fourth Useful Abstraction: Coarse-grain linear algebra objects

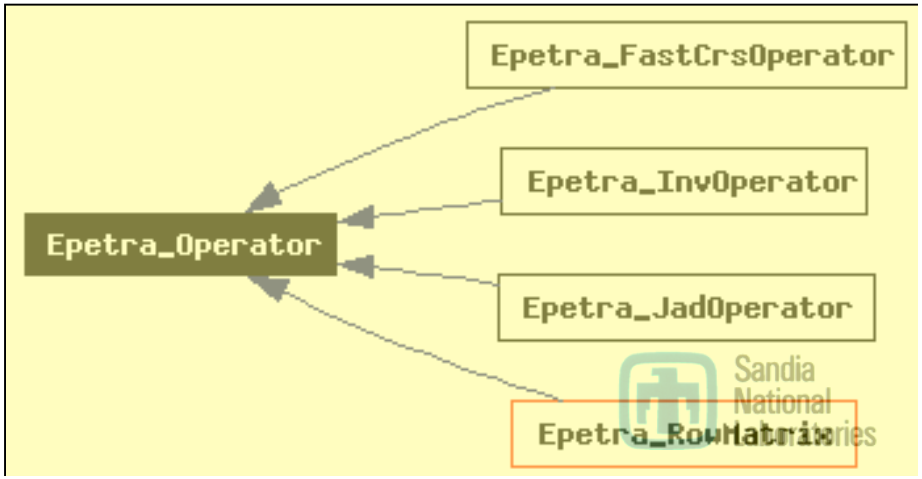
Sandia National Laboratories



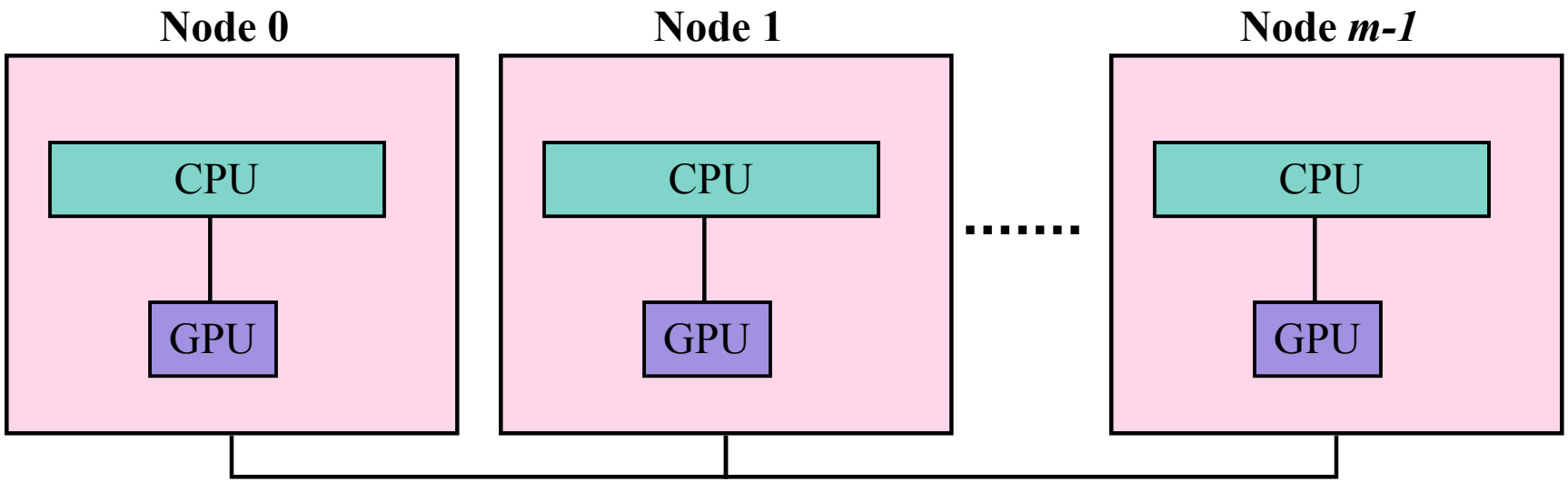
Epetra_Operator and Epetra_RowMatrix

Notes:

- **All Trilinos solvers can access linear operators via:**
  - **Epetra_Operator if no coefficients needed.**
  - **Epetra_RowMatrix if coefficients.**
  - **All Epetra matrix and operator class implement these two interfaces.**



GPU Utilization: $y = Ax$



- Epetra_GpuOperator:
  - Object data mostly resides on GPU.
  - Remains resident through many uses.
- Vectors:
  - Input vector x comes in for apply operation.
  - Output vector y is returned.

Sandia National Laboratories

Final Topic: Nested Parallelism

Sandia National Laboratories

Nested Parallelism

- Question: How to program for multi-core nodes?
- Similar to 1995: How do we program multi-processor nodes?
- ’95 Experience:
  - Hybrid programming model (threads/OpenMP under MPI) is complex.
  - Restructuring large-scale app for hybrid approach akin to serial-to-MPI conversion.
- ’95 Conclusion:
  - Just use MPI.
  - Must be SMP-Aware
  - Don’t ask for all procs on node.
- What’s different today?

Sandia National Laboratories

Summary

- Abstract machine model provides flexibility:
  - Required for Portability.
  - Enables state retention.
  - Allows specialized extensibility.
- Abstraction of distribution is dssential:
  - Required for efficient loop execution.
  - Supports easy construction of compatibly distributed objects.
  - Enables efficient redistribution mechanisms.
- Coarse-grain abstraction enables use of a range of computer systems:
  - (Within SPMD model) abstractions like Epetra_Operator seem to give us a fighting chance on incremental and novel architectures.
- (In the solvers area) Nested Parallelism is a Dubious Option:
  - Been there, done that (circa 95-96).
  - Limited opportunity for improvement.
  - Many opportunities to hinder performance.
  - Could (potentially) help with load imbalance.

Sandia National Laboratories